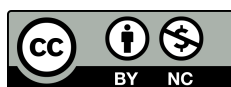




Apostila de JavaScript (ES6+)

Nível básico



Prof. Fausto G. Cintra
(professor@faustocintra.com.br)

03/07/2021

Sumário

APRESENTAÇÃO	2
1 COMEÇANDO COM JAVASCRIPT	4
1.1 Uma breve história do JavaScript (sim, é importante!)	4
1.2 Características da linguagem JavaScript	6
1.3 O JavaScript e as páginas Web	7
1.4 A estrutura de uma página HTML	7
1.5 Adicionando JavaScript a uma página HTML	8
1.6 Usando o console JavaScript	11
2 VARIÁVEIS E TIPOS DE DADOS	13
2.1 Declaração de variáveis	13
2.2 Atribuindo valores a variáveis	15
2.3 Tipos de dados	16
3 OPERADORES	19
3.1 Operadores aritméticos	19
4 ENTRADA E SAÍDA	26
5 ESTRUTURAS DE CONTROLE	27
6 ESTRUTURAS DE REPETIÇÃO	28
7 FUNÇÕES	29
8 DOCUMENT OBJECT MODEL (DOM)	30
9 VETORES	31
10 OBJETOS	32
11 MANIPULAÇÃO DE STRINGS	33
12 REVISITANDO O DOM	34

APRESENTAÇÃO

JavaScript é uma das linguagens de programação mais difundidas atualmente, sendo imprescindível para aplicações voltadas à Internet. Por isso, todos quantos queiram se tornar desenvolvedores Web precisam dominá-la.

Meu desejo, ao disponibilizar a você esta apostila, é ajudá-lo(a) a dar seus primeiros passos com a linguagem, com um conteúdo atualizado que abrange as principais modificações desde a versão ES6 (lançada em 2015). Este material é de nível básico; portanto, não cobre todos os recursos do JavaScript nem pretende ser a última palavra sobre o assunto. Sempre que algum tópico for secundário para quem está começando, ou exigir algum aprofundamento que saia da proposta do texto, há *links* para fontes externas.

A quem se destina esta apostila

Para o melhor aproveitamento deste material, é desejável que o estudante já tenha noções de:

- algoritmos e lógica de programação;
- arquitetura e organização de computadores; e
- uso de editores de código.

A quem NÃO se destina esta apostila

Se você ainda não tem nenhum conhecimento de programação, peço minhas desculpas, mas esta apostila não é para você. Existem alguns passos que você deve percorrer antes de se aventurar com JavaScript. Recomendo os seguintes materiais de estudo:

- [Curso em Vídeo: Algoritmos](#)
- [IF Fluminense: Apostila de Lógica de Programação](#)
- [Livro: Algoritmos: Lógica Para Desenvolvimento de Programação de Computadores - Edição Revisada e Atualizada \(Manzano e Oliveira\)](#)

Meus melhores votos de bons estudos!

Um abraço,

Prof. Me. Fausto G. Cintra

- professor@faustocintra.com.br
 - [Currículo Lattes](#)
-

Capítulo 1

COMEÇANDO COM JAVASCRIPT

1.1 Uma breve história do JavaScript (sim, é importante!)

Em 1989, **Tim Berners-Lee** inventou a linguagem HTML e, com ela, a possibilidade de criar páginas Web. Isso revolucionou a Internet da época, que, até então, era acessada principalmente por terminais de texto, com suporte muito limitado a imagens e outros tipos de mídia. Essa “nova” internet passou a ser conhecida pela sigla WWW (de *World Wide Web*, Teia de Alcance Mundial).

As páginas Web de Berners-Lee fizeram muito sucesso. Mas elas tinham uma limitação: apenas entregavam conteúdo multimídia, de forma estática, como a televisão. Não havia forma de o usuário interagir com esse conteúdo, nem de enviar informações de volta à origem. E isso restringia as possibilidades de uso das páginas Web.

Em 1994, o **Netscape Navigator** era o navegador Web líder de mercado. Para resolver o problema da falta de interatividade, a empresa Netscape decidiu por introduzir uma linguagem de programação que pudesse ser incorporada às páginas Web e executada pelo navegador ao carregá-las.

A linguagem em questão, depois de muitos nomes provisórios, ganhou o nome de JavaScript, tendo sido lançada em 1995. Foi desenvolvida originalmente por **Brendan Eich**, que, nos dias de hoje, é o nome por trás do navegador **Brave**, focado em privacidade e eliminação de propagandas indesejadas.

A escolha pelo nome JavaScript não foi por acaso, tratando-se, na verdade, de uma bela jogada de *marketing*. Na época do lançamento, uma outra linguagem, também recém-surgida, estava fazendo bastante sucesso: o **Java**, da empresa Sun Microsystems. A intenção da Netscape era embarcar na popularidade do Java, mas, no fim das contas, acabou por causar uma grande confusão, levando muitos a acreditar que o JavaScript

era baseado em Java. No entanto, **JavaScript e Java são linguagens totalmente diferentes**, tanto em finalidade quanto na forma de se programar e, ainda hoje, os mais desavisados pensam que se trata da mesma coisa por causa da semelhança do nome.

A Microsoft, uma das gigantes da tecnologia já naquela época, não podia ficar de fora da onda da WWW. Em 1996, a empresa lançou o **Internet Explorer 3.0** com sua própria “versão” da linguagem JavaScript, a qual denominou JScript. O JScript possuía várias extensões e diferenças em relação ao JavaScript original, fazendo com que *websites* feitos com ele só funcionassem corretamente no Internet Explorer.

Diante desse fato, em novembro de 1996, a Netscape anunciou que havia submetido o JavaScript à **ECMA** (*European Computer Manufacturers Association*, ou Associação Europeia de Fabricantes de Computadores) como candidata a padrão industrial. Com isso, a empresa abriu as especificações da linguagem para o público, como se quisesse dizer que “não tinha nada a esconder”, ao contrário da concorrente. O trabalho que se seguiu resultou na versão padronizada chamada **ECMAScript** (ECMA-262), versão 1.0 (junho de 1997). O padrão foi evoluindo ao longo dos anos, sendo a versão mais recente a ES2020 (ECMAScript 2020), lançada em junho de 2020. Atualmente, o ECMAScript é considerado o padrão de JavaScript a que todo navegador deve dar suporte.

Contudo, a chamada Primeira Guerra dos *Browsers* (~ 1995—2001) não demorou a começar. A Microsoft passou a fornecer seu navegador junto com o sistema operacional Windows, fazendo com que as pessoas não mais precisassem recorrer ao *download* do Netscape para ter um programa do tipo. Durante praticamente toda a primeira década do século, o Internet Explorer reinou absoluto na liderança do mercado de navegadores. Com a posição alcançada, a Microsoft ignorava o padrão ECMAScript, investindo em suas extensões proprietárias. Desenvolver para a Web tornou-se caótico, devido à falta de compatibilidade entre o JScript do Internet Explorer com o JavaScript padrão.

A empresa Netscape fechou as portas em 2002, exaurida por uma longa batalha judicial contra a Microsoft. Dos escombros da Netscape surgiram dois projetos, que resultaram nos navegadores **Opera** e **Firefox**, que existem até hoje. Embora fossem bons produtos e tivessem alcançado algum destaque, nenhum deles tinha força para competir com a toda poderosa Microsoft.

Foi necessário que outra gigante da tecnologia entrasse no páreo para ameaçar a liderança da Microsoft e do Internet Explorer entre os navegadores, iniciando a Segunda Guerra dos *Browsers*. Em 2008, a Google lançou o **Google Chrome**, com importantes diferenciais. Seu mecanismo de processamento de JavaScript, chamado V8, era extremamente rápido, fazendo com que as páginas Web carregassem bem mais depressa que no Internet Explorer. Além disso, o Chrome era baseado em um projeto de código aberto chamado **Chromium**, o que incentivou desenvolvedores a colaborar para o aper-

feiçãoamento do produto. Mais do que isso, a Google adotou como princípio a aderência aos chamados “padrões Web”, dentre os quais se encontra o ECMAScript.

Por volta de 2013, cinco anos após seu lançamento, o Google Chrome conquistou a liderança do mercado de navegadores. A Microsoft se viu obrigada a substituir o Internet Explorer por um novo navegador, o Edge, mais conforme aos padrões Web.

Antes, em 2010, aproveitando-se do fato de o Chrome ter seu código-fonte aberto, Ryan Dahl isolou o código do interpretador de JavaScript V8 e criou o **Node.js**. Este *software* permite com que o JavaScript, uma linguagem de Internet projetada para ser executada dentro do navegador, possa ser executado também fora dele, tornando-a uma linguagem de propósito geral.

A aderência do Google aos padrões Web, a derrocada do Internet Explorer e seu JScript não-padrão e a popularidade do Node.js levaram a um impulsionamento do desenvolvimento da linguagem JavaScript, que vem ganhando uma nova versão por ano desde 2015, quando foi lançada a versão ES6.

A versão ES6 representou um marco de modernização na linguagem, e as versões subsequentes prosseguiram com a adição de funcionalidades. **Esta apostila tem como um de seus objetivos ensinar o JavaScript moderno**, abrangendo as principais características introduzidas desde o ES6, sem deixar de mencionar as formas originais, quando existirem.

1.2 Características da linguagem JavaScript

Dentre os principais atributos da linguagem, podemos destacar:

- ela é **interpretada**, ou seja cada linha de código é transformada em linguagem de máquina à medida que vai sendo executada pelo navegador. Existe um outro tipo de linguagem, as compiladas, em que todo o código de um programa é transformado em linguagem de máquina para só então poder ser executado.
- ela é **imperativa**: você deve fornecer instruções claras de como conseguir o que quer, assim como em Python, VBA e C, entre outras linguagens. Já nas linguagens do tipo declarativo, é necessário dizer apenas o que se quer, sem precisar indicar as instruções passo a passo.
- ela é **estruturada**, possuindo estruturas de construção de blocos lógicos de código, como a grande maioria das linguagens de programação utilizadas na atualidade. Sua sintaxe (forma de escrever as instruções) assemelha-se à das linguagens C e PHP.
- ela tem **tipagem dinâmica**: não é necessário indicar o tipo ao declarar uma variável, sendo ele determinado a partir do valor armazenado na variável. Como

consequência, uma mesma variável pode inicialmente guardar um número e, mais tarde, ter seu valor trocado por uma *string*.

Caso você tenha tido contato com JavaScript antes, principalmente se há mais de cinco anos, pode se lembrar de que era costume colocar um ; (ponto-e-vírgula) ao final de cada linha de código, como terminador de instruções. Saiba que esses **ponto-e-vírgulas finais são opcionais** em JavaScript (exceto em raríssimos casos), e, portanto, você não os verá nesta apostila.

1.3 O JavaScript e as páginas Web

As páginas Web são formadas pela combinação de três tecnologias:

- **HTML** (*Hypertext Markup Language*, isto é, linguagem de marcação de hipertexto): é responsável por estruturar o conteúdo da página, ou seja, **o que** você vê nela.
- **CSS** (*Cascading Style Sheets*, folhas de estilo em cascata): cuida da aparência da página, como cores, fontes e alinhamento dos elementos. Controla **como** o conteúdo é exibido na página.
- **JavaScript**: é uma linguagem de **programação** que pode ser adicionada às páginas Web, conferindo-lhes interatividade para com o usuário. O JavaScript é capaz, por exemplo, de verificar em um formulário se o usuário digitou algo diferente do esperado ou de fazer coisas se movimentarem pela página.

IMPORTANTE

Como o foco desta apostila é a linguagem JavaScript, veremos apenas o necessário de HTML e CSS, sem aprofundamento. Caso você queira aprender mais sobre essas duas tecnologias, recomendo o [Curso em Vídeo de HTML e CSS](#).

A linguagem JavaScript será o objeto do nosso estudo. No entanto, como iremos utilizá-la **dentro** de uma página Web, é necessário aprender o básico de HTML.

1.4 A estrutura de uma página HTML

Uma página Web (também chamada de página HTML) é um arquivo de texto simples com extensão `.html` ou `.htm`. A Listagem 1.1 seguir apresenta a estrutura básica de um arquivo HTML.

Listagem 1.1 Estrutura básica de um arquivo HTML

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="UTF-8">
5   <title>Minha primeira página HTML</title>
6 </head>
7 <body>
8   <p>Olá, mundo!</p>
9 </body>
10</html>
```

IMPORTANTE

Os números que aparecem à esquerda do código **não** fazem parte dele. Servem apenas para que possamos nos referir a diferentes partes do código usando o número da linha.

Como podemos ver, o HTML é formado por elementos delimitados pelos caracteres < e >, os quais são chamados **tags**.

Muitas *tags* vêm em pares, formando **seções**. Podemos observar, por exemplo, que a tag <body>, chamada *tag* de abertura, tem a correspondente *tag* de fechamento </body> (note a presença da / antes do nome da *tag*).

Agora, vamos analisar cada uma das partes desse código.

- <!DOCTYPE html> (linha 1): essa *tag* serve para indicar ao navegador Web que irá exibir a página qual a versão da linguagem HTML está sendo usada. No caso, esse *doctype* indica que se trata da versão 5 do HTML, a mais recente.
- Seção **html** (linhas 2 a 10): a maior parte do código da página fica nessa grande seção. Dentro dessa grande seção, temos as seções **head** e **body**.
- Seção **head** (linhas 3 a 6): aqui colocadas *tags* de configuração da página, como a <meta charset="UTF-8"> (linha 4), para garantir que os caracteres acentuados sejam exibidos corretamente. Os elementos dessa seção, normalmente, não têm um efeito visível para o usuário. Uma exceção é a *tag* <title> (linha 5), cujo conteúdo aparece na aba do navegador onde a página estiver sendo exibida.
- Seção **body** (linhas 7 a 9): todo o conteúdo da página que será visível para o usuário é colocado nessa seção. No código de exemplo, temos um parágrafo (<p>, linha 8) contendo um texto a ser exibido.

Quando o arquivo HTML contendo este código for exibido em um navegador Web, veremos um resultado semelhante ao da Figura 1.1:

1.5 Adicionando JavaScript a uma página HTML

Para utilizar JavaScript em uma página HTML, precisamos criar uma seção

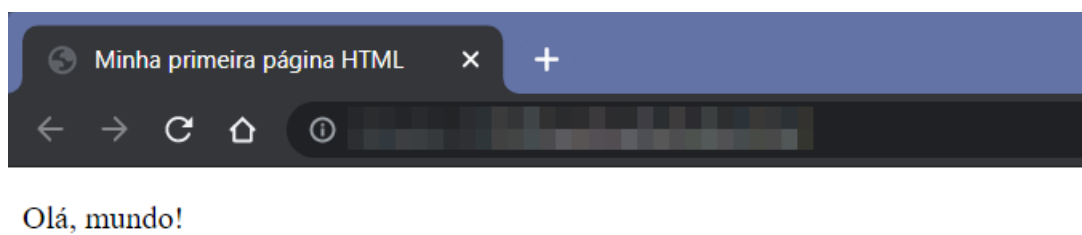


Figura 1.1: Exibição de uma página HTML

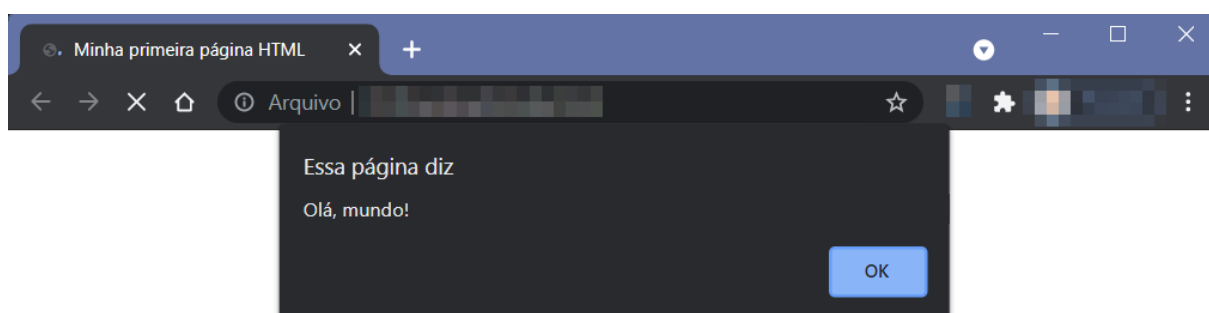


Figura 1.2: Mensagem exibida usando JavaScript

Listagem 1.2 Código JavaScript em uma página HTML

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="UTF-8">
5   <title>Minha primeira página HTML</title>
6   <script>
7     alert('Olá, mundo!')
8   </script>
9 </head>
10 <body>
11
12 </body>
13 </html>
```

Portanto, agora você já sabe. Toda vez que formos usar JavaScript em uma página Web, devemos:

1. Criar um arquivo com extensão `.html` ou `.htm`.
2. Colocar dentro desse arquivo a estrutura básica de um arquivo HTML. Editores de código, como o **Visual Studio Code** ou o **Gitpod** possuem recursos que geram automaticamente este código.
3. Adicionar uma seção `<script></script>` na seção **head** e colocar as instruções JavaScript dentro dela.

1.5.1 Adicionando comentários

Bons desenvolvedores não se preocupam apenas em escrever bem as instruções que serão executadas. Eles também cuidam da documentação, adicionando comentários que explicam os principais pontos, como forma de colaborar para que o código seja compreendido por outras pessoas (e até pelo autor, no futuro).

Todas as linguagens possuem maneiras de inserir comentários no código. Em JavaScript, eles assumem duas formas:

- **comentários de linha:** são iniciados com `//` (duas barras) e, como o próprio nome indica, terminam junto com o fim da linha onde foram colocados.
- **comentários de bloco:** começam com `/*` (barra asterisco) e terminam com `*/` (asterisco barra). Tudo o que estiver entre eles é considerado comentário, que pode ter várias linhas.

A Listagem 1.3 exemplifica esses tipos de comentários.

Listagem 1.3 Comentários em JavaScript

```
1 // Exibe uma mensagem em uma caixa de diálogo
2 alert('Olá, como vão seus estudos?') // Posso comentar aqui tb
3
4 /*
5     A linha abaixo exibe um texto na área <body>
6     do arquivo HTML
7 */
8 document.write('Estou adorando aprender JavaScript!')
```

IMPORTANTE

A partir desta listagem, mostraremos apenas o código JavaScript. Você já sabe que ele precisa estar dentro das tags `<script></script>` do arquivo HTML, não é mesmo? ;)

1.6 Usando o console JavaScript

Todos os navegadores mais utilizados atualmente tem uma parte “secreta”, desconhecida da maioria dos usuários. Essa parte é chamada de Ferramentas de Desenvolvedor e pode ser acessada ao pressionar a tecla F12. Será aberto um painel, no lado direito ou inferior da tela, conforme mostrado na Figura 1.3.

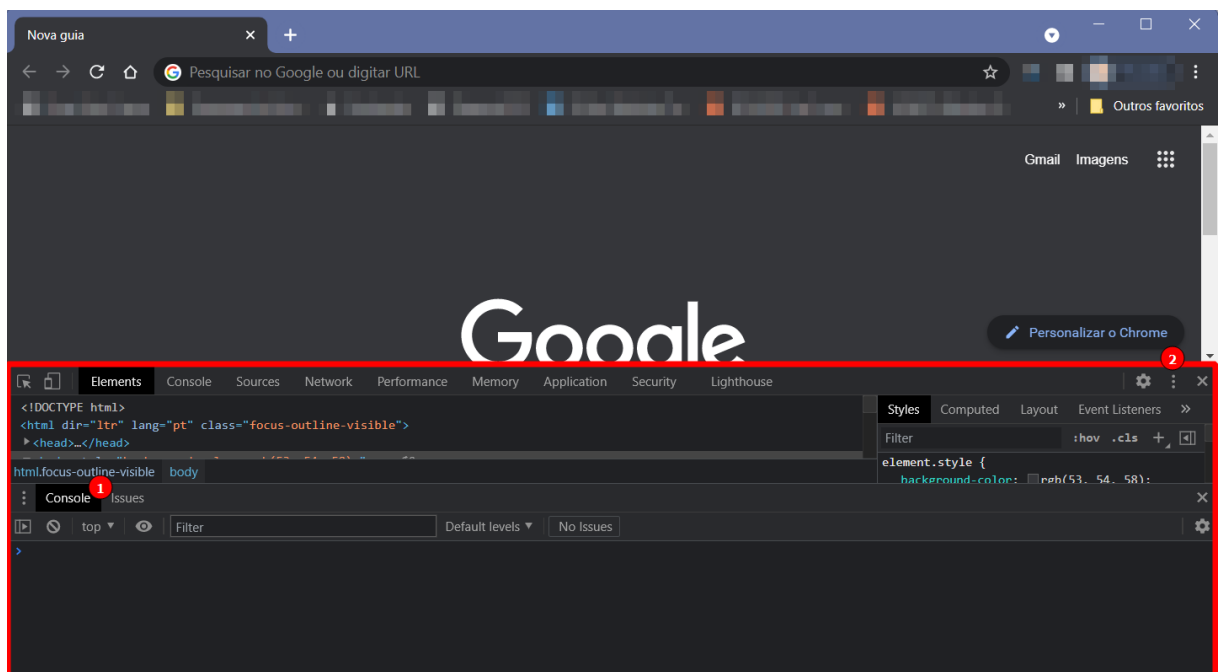


Figura 1.3: Navegador Web exibindo as Ferramentas de Desenvolvedor (no destaque). Note (1) a aba Console e (2) o menu de opções

DICA: usando o menu de opções (2), é possível mudar o posicionamento das Ferramentas de Desenvolvedor na tela.

Na aba Console, é possível digitar instruções JavaScript, incluindo operações aritméticas, e ver imediatamente seu resultado. Veja alguns exemplos na Figura 1.4.

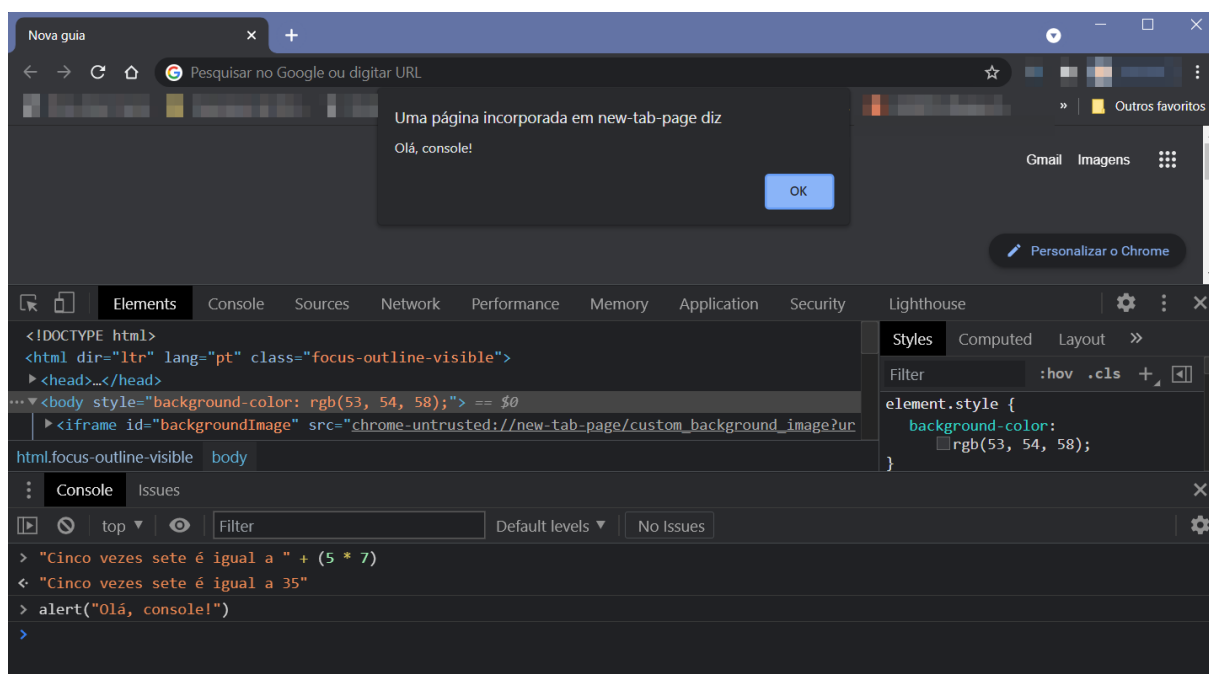


Figura 1.4: Resultado da execução de alguns comandos no console JavaScript do navegador Web

Capítulo 2

VARIÁVEIS E TIPOS DE DADOS

2.1 Declaração de variáveis

O Javascript é uma linguagem de tipagem dinâmica. Isso significa que os tipos de dados de suas variáveis não são determinados no momento em que são declaradas. Em vez disso, eles são deduzidos a partir dos **valores** atribuídos a elas.

Veja alguns exemplos na Listagem 2.1:

Listagem 2.1 Declarações de variáveis em JavaScript

```
1 let x
2 var preco
3 const meuNome = 'Fausto'
```

NOTE BEM: para declarar uma variável em JavaScript, basta uma das palavras-chave reservadas (**let** ou **var**) seguida do nome da variável, nada mais. **const** exige também que um valor seja atribuído à variável no momento da declaração.

Há três palavras-chave utilizadas para declarar variáveis em JavaScript:

- **let**: atualmente, é o método recomendado de criação de variáveis. Uma das vantagens de sua utilização é a impossibilidade de se declarar mais de uma variável com o mesmo nome, o que ajuda a evitar erros de lógica no código. Possui também outros benefícios e características que serão explicadas ao longo desta apostila. É uma adição relativamente recente à linguagem (foi introduzida na versão ES6, de 2015).
- **var**: é a palavra-chave originalmente disponível para a declaração de variáveis, desde a primeira versão do JavaScript. Seu uso apresenta alguns problemas, como a possibilidade de redeclarar uma variável já existente, o que pode induzir a

erros de lógica. Evite utilizá-la, até compreender completamente as consequências de seu emprego.

- **const**: em algumas situações, é necessário representar valores que não devem mais ser alterados posteriormente. São as chamadas **constantes**. Variáveis declaradas com **const** devem receber um valor quando declaradas e não aceitam que este valor seja modificado depois.

Várias variáveis podem ser declaradas simultaneamente (Listagem 2.2):

Listagem 2.2 Declarações múltiplas de variáveis

```
1 let quantidade, precoUnitario, precoTotal
2 let nome, email, telefone, celular
```

2.1.1 Nomeando variáveis

Em JavaScript, nomes de variáveis devem começar com uma letra; os caracteres \$ e _ também são aceitos primeira posição. Dígitos (0 a 9) podem ser utilizados a partir da segunda posição.

Você também não pode usar para nomear variáveis nenhuma das **palavras reservadas** pelo JavaScript para o seu próprio uso.

É importante frisar que JavaScript é uma **linguagem sensível à diferença entre letras maiúsculas e minúsculas** (*case sensitive*, em inglês), isto é, ela trata letras maiúsculas e minúsculas como coisas diferentes. Assim, uma variável de nome `num` é diferente de outra variável de nome `NUM`, e ambas são diferentes de uma terceira variável nomeada como `Num`.

Apesar de as especificações da linguagem JavaScript assim permitirem, não é recomendável declarar variáveis que contenham caracteres acentuados.

Listagem 2.3 Exemplos de nomeação de variáveis

```
1 let x           // OK!
2 let primeiroNome // OK!
3 let 1nome      // INVÁLIDO: começa com um dígito
4 let $valor     // OK, mas pouco usual
5 let _num       // OK, mas pouco usual
6 let %resultado // Caractere inicial INVÁLIDO
7 let área       // OK, mas acentos não são recomendados
```

2.1.2 Convenções de nomeação de variáveis

Quando muitos desenvolvedores trabalham num mesmo projeto, é comum que surja, mais cedo ou mais tarde, alguma discórdia sobre a forma de nomear as variáveis.

Por isso, as comunidades de cada linguagem acabam adotando convenções, que não são regras definidas na própria linguagem, mas sim uma espécie de “combinado” entre seus membros.

A convenção mais comum entre os desenvolvedores JavaScript é a seguinte:

1. **Sempre iniciar** o nome das variáveis com uma **letra minúscula**; e
2. Se o nome da variável for composto por **mais de uma palavra**, é utilizada **inicial maiúscula a partir da segunda** palavra.

Observe os exemplos da Listagem 2.4:

Listagem 2.4 Uso da convenção 'camel case' na nomeação de variáveis

```
1 // Uma palavra, inicial minúscula
2 let area
3 // Duas palavras, a segunda com inicial maiúscula
4 let areaTerreno
5 // Três palavras, iniciais maiúsculas a partir da segunda
6 let areaTerrenoPadrao
```

CURIOSIDADE: esse tipo de convenção é chamado, em inglês, de *camel case* (*camel* significa “camelo”). O motivo é que as letras maiúsculas no meio do nome das variáveis acabam se parecendo com as corcovas do animal.

2.2 Atribuindo valores a variáveis

Para atribuir valor a uma variável, é usado o operador = em JavaScript. A Listagem 2.5 exemplifica os diferentes formas de atribuição.

Listagem 2.5 Exemplos de atribuição de valores a variáveis

```
1 let quantidade, valor
2
3 quantidade = 7
4 valor = 12.63
5
6 // Podemos atribuir um valor à variável quando a declaramos
7 let cargo = 'Gerente'
8
9 /* Podemos, inclusive, fazer várias declarações/atribuições
10 de uma só vez */
11 let marca = 'Volkswagen', modelo = 'Fusca', ano = 1969
```

A atribuição de valores pode ocorrer posteriormente à declaração da variável (linhas 3 e 4) ou acontecer ao mesmo tempo que esta (linhas 7 e 10). Neste último caso, dizemos que a variável, além de declarada, foi **inicializada**.

2.3 Tipos de dados

Vamos analisar em detalhes os **tipos de dados** disponíveis na linguagem JavaScript. Usaremos, como referência e exemplo, a Listagem 2.6.

Listagem 2.6 Exemplos de tipos de dados

```
1  let nome, sobrenome, naturalidade, idade, altura, peso, casado
2  let conjugue, ocupacao, filhos, nomeCompleto
3  nome = "Afrânio" // string
4  sobrenome = 'Azeredo' // string
5  naturalidade = `Morro Alto de Cima (MG)` // string
6  idade = 44 // number
7  altura = 1.77 // number
8  peso = undefined // undefined
9  casado = true // boolean
10 conjugue = { nome: 'Jeruza', sobrenome: 'Jordão' } // object
11 ocupacao = null // object
12 filhos = ['Zózimo', 'Zuleica'] // object
13 nomeCompleto = function(nome, sobrenome) { // function
14     return nome + " " + sobrenome
15 }
```

- **string** (linhas 3 a 5): representa uma sequência de caracteres, ou seja, um texto. Em JavaScript, *strings* podem ser delimitadas com aspas duplas ("), aspas simples (') ou acentos graves (`), muitas vezes chamados também de crases. Aspas simples e aspas duplas são totalmente equivalentes entre si, e a escolha por uma ou por outra acaba sendo decisão do programador. Já as *strings* delimitadas por acentos graves têm significado e funções especiais, que serão explicadas mais à frente.
- **number** (linhas 6 e 7): ao contrário de outras linguagens de programação, o JavaScript não faz distinção entre números inteiros e números com parte fracionária (também chamados de números de ponto flutuante), colocando-os todos sob um mesmo tipo. Para separar a parte inteira da parte fracionária, quando esta exista, é sempre usado o ponto (.), embora, na língua portuguesa, usemos a vírgula para esse fim. Existem várias outras formas de representar valores numéricos:
 - valores hexadecimais (base 16) podem ser representados usando-se o prefixo 0x. Por exemplo 0x1A representa o valor hexadecimal 1A (equivalente a 26 em decimal).
 - valores octais (base 8) são representados usando-se um 0 no início. **CUIDADO!** Para o JavaScript, 045 não é um 45 decimal com um inútil zero à esquerda, e sim o valor octal 45 (que, convertido em decimais, equivale a 37).
 - Números muito grandes ou muito pequenos podem usar a chamada notação

científica. Assim, $4e12$ é o mesmo que 4 vezes 1 seguido de doze zeros, ou seja 4000000000000.

- Para números *realmente* grandes, foi introduzida na versão 2020 do EcmaScript mais um tipo de dados, o **bigint**. Você pode saber detalhes dessa novidade consultado a documentação da [Mozilla Developer Network](#).
- **undefined** (linha 8): é um tipo especial, usado para indicar que uma variável não tem qualquer valor atribuído a ela. A propósito, toda variável declarada e que ainda não recebeu um valor é considerada **undefined**.
- **boolean** (linha 9): como indicado pelo próprio nome, representa valores booleanos. Os únicos valores possíveis são **true** (verdadeiro) e **false** (falso), sempre escritos totalmente em minúsculas.
- **object** (linhas 10 a 12): é o tipo de dados mais versátil da linguagem, usado, normalmente, para armazenar vários valores em uma única variável. Os vetores (linha 12) e objetos em sentido estrito (linha 10) fazem parte desta categoria. **null** é um valor especial utilizado para representar um objeto inexistente.
- **function** (linhas 13 a 15): em JavaScript, funções podem ser atribuídas a variáveis. Essa característica é importante para usos de nível intermediário e avançado da linguagem.

2.3.1 Descobrindo o tipo do valor de uma variável

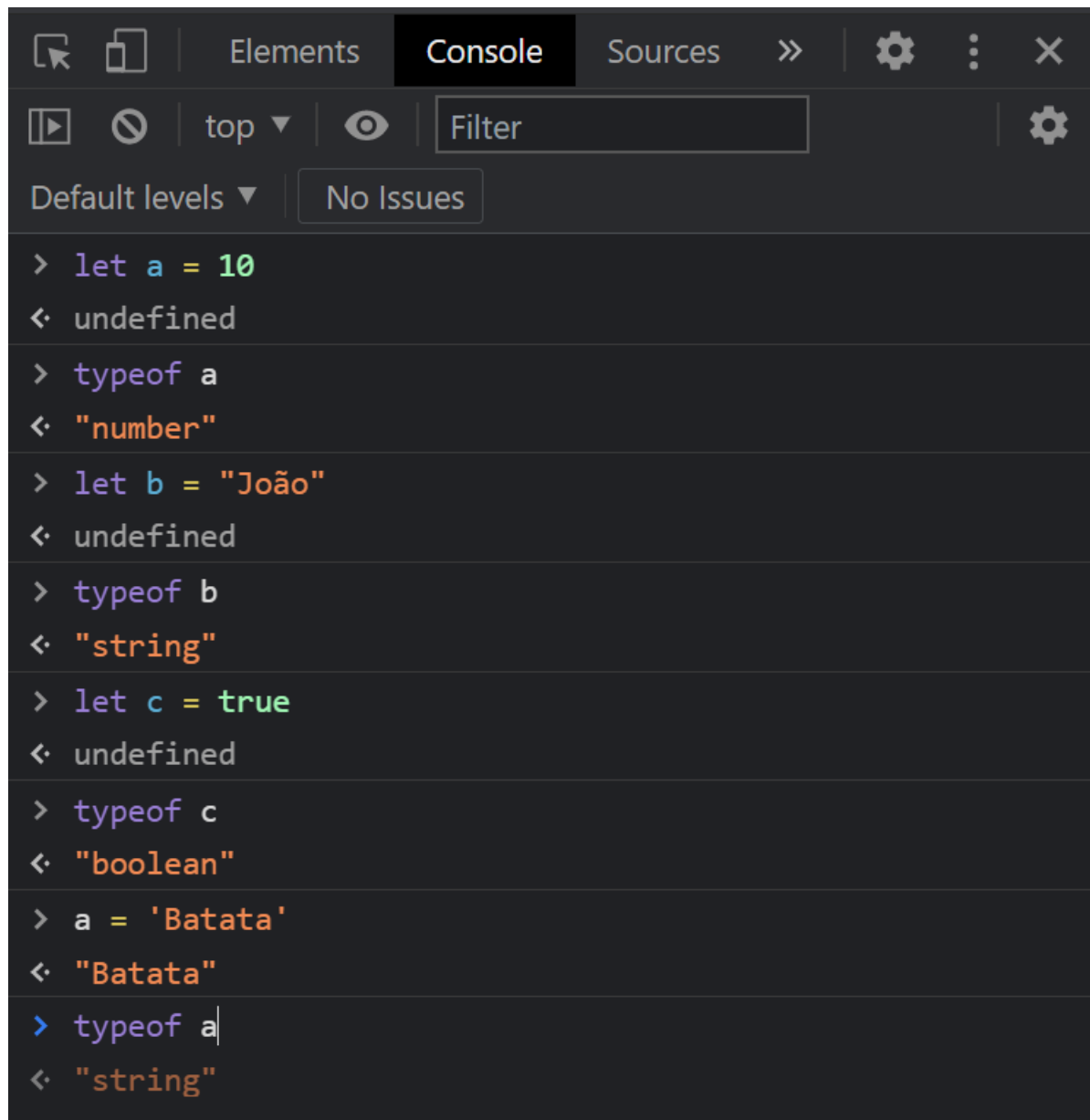
Em JavaScript, embora as variáveis não tenham um tipo determinado, os valores que elas abrigam têm. Para descobrir qual o tipo de dados do valor em um dado momento, usamos o operador **typeof**.

Observe os testes feitos no console JavaScript mostrados na Figura 2.1.

OBSERVAÇÃO: repare que as instruções que inicializam variáveis (começadas pela palavra **let**) retornaram **undefined**. Isso é normal, e significa que a instrução de inicialização da variável não retorna valor algum.

O operador **typeof** nos diz qual o tipo de dados que a variável possui **no momento do teste**. Veja o que aconteceu com a variável `a`. No primeiro teste, obtemos `"number"`, quando o valor que ela tinha era `10`. Mais à frente, depois que mudamos o valor de `a` para `"Batata"`, o teste retornou o tipo do novo valor, `"string"`. Com isso, não resta dúvida: **o que possui tipo é o valor da variável, não a variável em si mesma**.

Agora que sabemos como funcionam as variáveis e tipos de dados em JavaScript, precisamos saber o que fazer com eles. Bora aprender operadores?



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays the following code and output:

```
> let a = 10
< undefined

> typeof a
< "number"

> let b = "João"
< undefined

> typeof b
< "string"

> let c = true
< undefined

> typeof c
< "boolean"

> a = 'Batata'
< "Batata"

> typeof a
< "string"
```

Figura 2.1: Determinando o tipo do valor de algumas variáveis no console JavaScript

Capítulo 3

OPERADORES

3.1 Operadores aritméticos

Convivemos com os operadores aritméticos das quatro operações básicas desde o Ensino Fundamental. A maior parte das pessoas não terá problemas com os operadores das quatro operações aritméticas básicas. Quem usa computador há algum tempo sabe que o caractere `*` é usado para multiplicação e o caractere `/` é empregado na divisão.

É assim também no JavaScript, como ilustra a Figura 3.1.

OBSERVAÇÃO: ao contrário da maioria das linguagens de programação (e até da sua calculadora), JavaScript não retorna erro quando há uma tentativa de divisão por zero. Em vez disso, ele retorna `Infinity` caso o dividendo seja positivo ou `-Infinity` se o dividendo for negativo. Existem razões matemáticas para tanto. Há uma bela discussão sobre isso [aqui](#).

Além deles, a linguagem conta com mais dois operadores aritméticos (Figura 3.2):

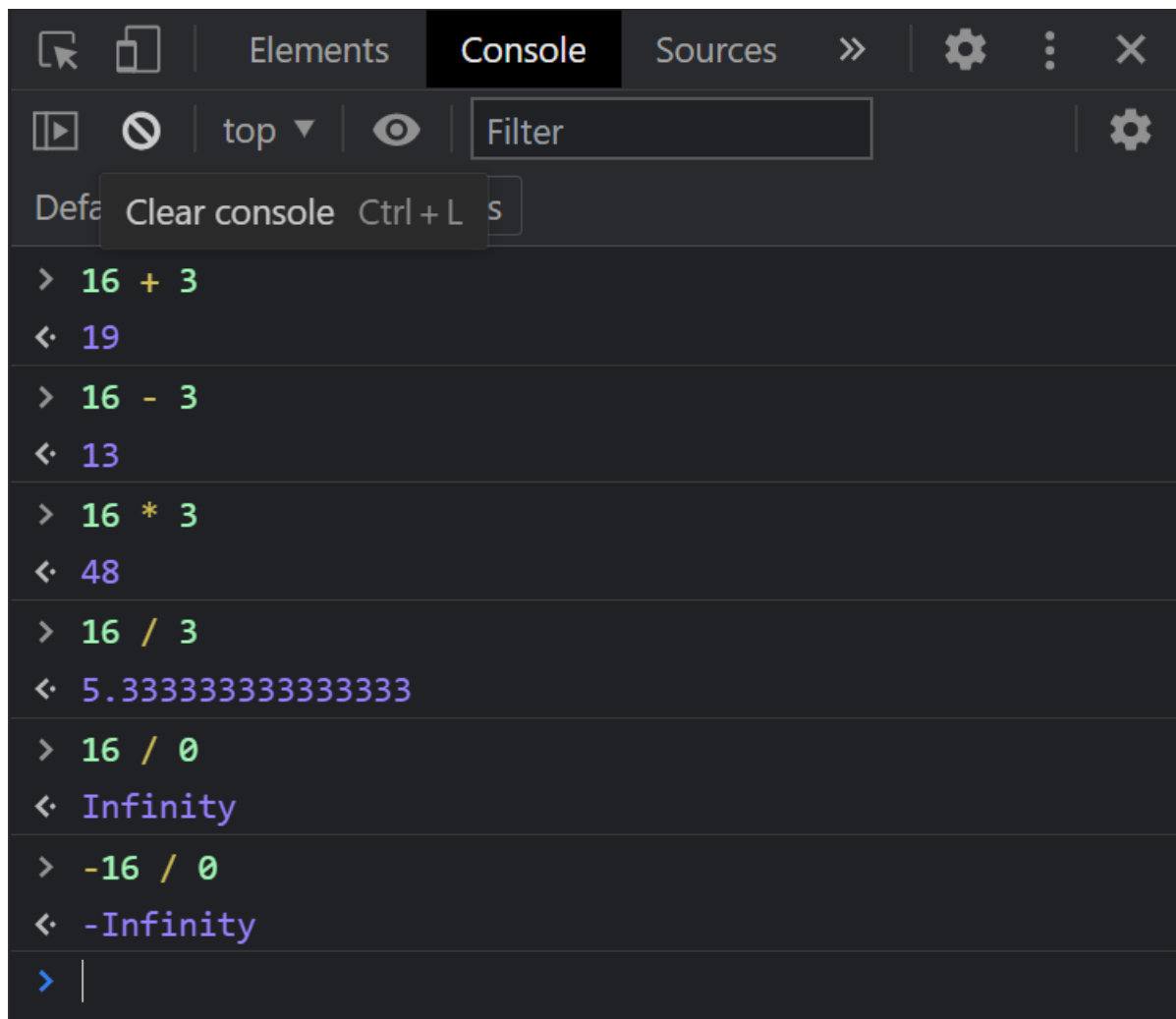
- `%`: é o operador de **resto da divisão**, também chamado de módulo da divisão. Calcula quanto sobra (o resto) da divisão de um número pelo outro.
- `**`: dois asteriscos consecutivos representam o operador de **potenciação**, que calcula o resultado do primeiro número elevado à potência do segundo.

3.1.1 Quando os operandos não são números

Os operadores aritméticos funcionam como esperado quando seus operandos são números. Mas, e quando não são? Aí **depende**.

Vamos fazer alguns testes com o operador de multiplicação (Figura 3.3).

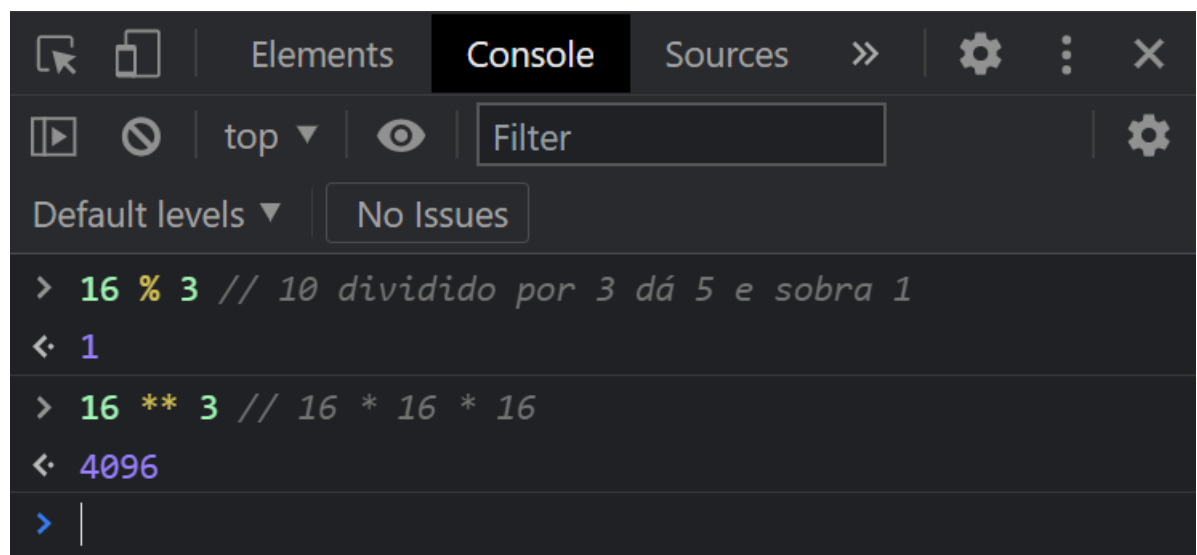
Sem dúvida, são resultados surpreendentes. Vamos analisar caso a caso:



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays a series of JavaScript arithmetic operations and their results. The operations are: `16 + 3` (result: 19), `16 - 3` (result: 13), `16 * 3` (result: 48), `16 / 3` (result: 5.333333333333333), `16 / 0` (result: Infinity), and `-16 / 0` (result: -Infinity). The console interface includes a 'Filter' input, a 'Clear console' button, and a 'Default levels' dropdown.

```
> 16 + 3
< 19
> 16 - 3
< 13
> 16 * 3
< 48
> 16 / 3
< 5.333333333333333
> 16 / 0
< Infinity
> -16 / 0
< -Infinity
> |
```

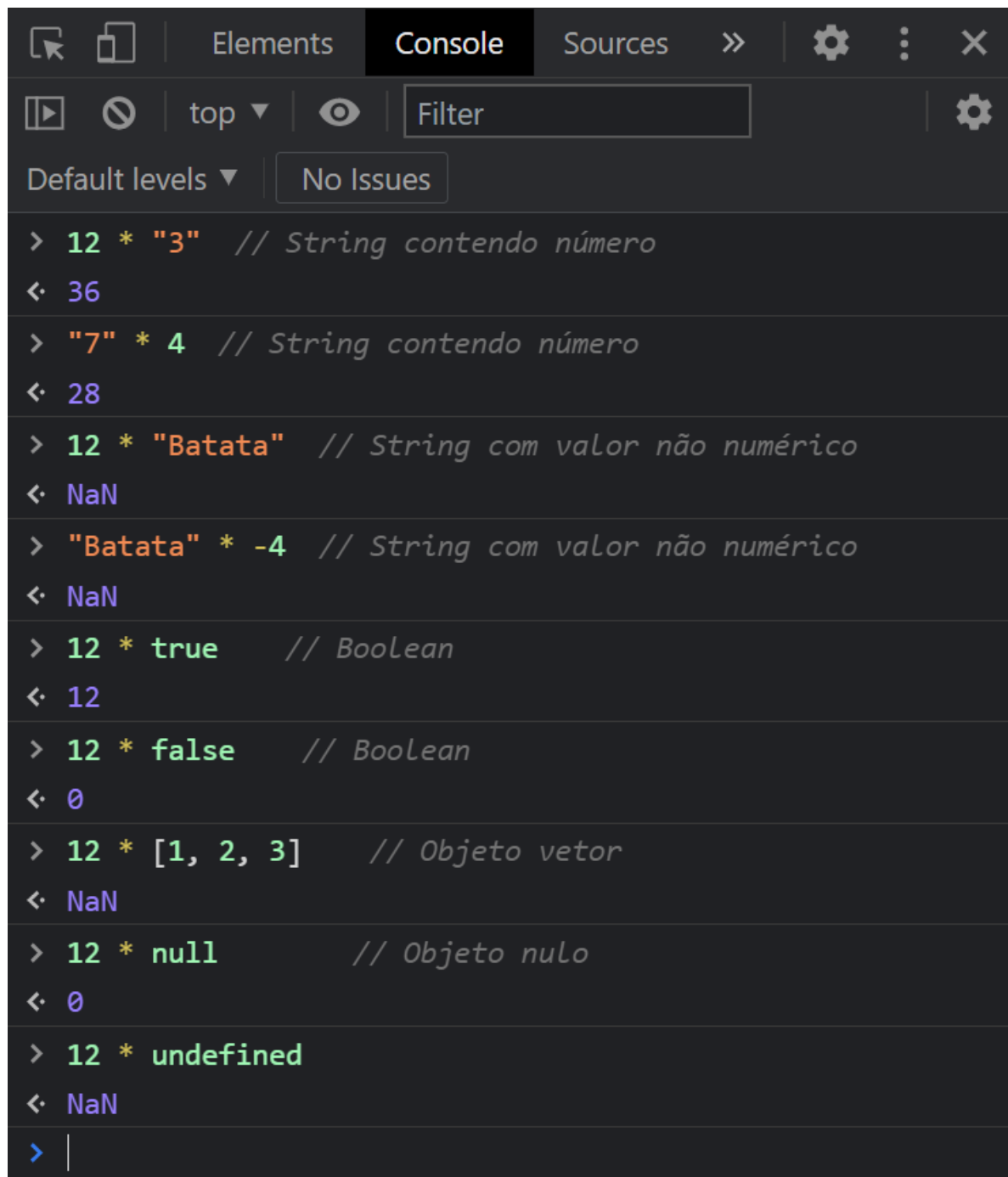
Figura 3.1: Os quatro operadores aritméticos básicos em JavaScript



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays two JavaScript operations: `16 % 3` (result: 1) with a comment `// 10 dividido por 3 dá 5 e sobra 1`, and `16 ** 3` (result: 4096) with a comment `// 16 * 16 * 16`. The console interface includes a 'Filter' input, a 'No Issues' button, and a 'Default levels' dropdown.

```
> 16 % 3 // 10 dividido por 3 dá 5 e sobra 1
< 1
> 16 ** 3 // 16 * 16 * 16
< 4096
> |
```

Figura 3.2: Os operadores de resto da divisão e de potenciação



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays the results of several JavaScript expressions involving multiplication with non-numeric operands. Each expression is followed by a comment in Portuguese explaining the operand type. The results are as follows:

- `12 * "3"` (String contendo número) results in `36`.
- `"7" * 4` (String contendo número) results in `28`.
- `12 * "Batata"` (String com valor não numérico) results in `NaN`.
- `"Batata" * -4` (String com valor não numérico) results in `NaN`.
- `12 * true` (Boolean) results in `12`.
- `12 * false` (Boolean) results in `0`.
- `12 * [1, 2, 3]` (Objeto vetor) results in `NaN`.
- `12 * null` (Objeto nulo) results in `0`.
- `12 * undefined` results in `NaN`.

```
> 12 * "3" // String contendo número
< 36
> "7" * 4 // String contendo número
< 28
> 12 * "Batata" // String com valor não numérico
< NaN
> "Batata" * -4 // String com valor não numérico
< NaN
> 12 * true // Boolean
< 12
> 12 * false // Boolean
< 0
> 12 * [1, 2, 3] // Objeto vetor
< NaN
> 12 * null // Objeto nulo
< 0
> 12 * undefined
< NaN
> |
```

Figura 3.3: Operandos não numéricos

1. Quando um dos operandos é *string*, temos duas possibilidades:
 - a) se o conteúdo da *string* equivaler a um valor numérico, o JavaScript efetua automaticamente a conversão de tipos e trata o valor da *string* como número, e temos o resultado da operação aritmética como se todos os operandos fossem numéricos.
 - b) se o conteúdo da *string* não contiver um valor que possa ser convertido para número, a operação aritmética é impossível e recebemos, para indicar esse fato, o valor especial NaN, que significa **Not a Number** (não é um número).
2. No caso de um dos operandos ser *boolean*, o valor **true** é tratado como se valesse 1 e o valor **false** é considerado como 0, e a operação aritmética é feita normalmente.
3. No caso de operandos do tipo *object*, não é possível efetuar a operação, portanto recebemos o resultado NaN. Uma exceção é o objeto nulo (**null**), que é tratado como 0 e possibilita a operação.

CONFIRA VOCÊ MESMO(A)

Resultados semelhantes são obtidos com os operadores de subtração (-), divisão (/), resto da divisão (%) e potenciação (**). Agora é a sua vez de abrir as Ferramentas do Desenvolvedor (tecle F12 no navegador) e fazer seus próprios testes.

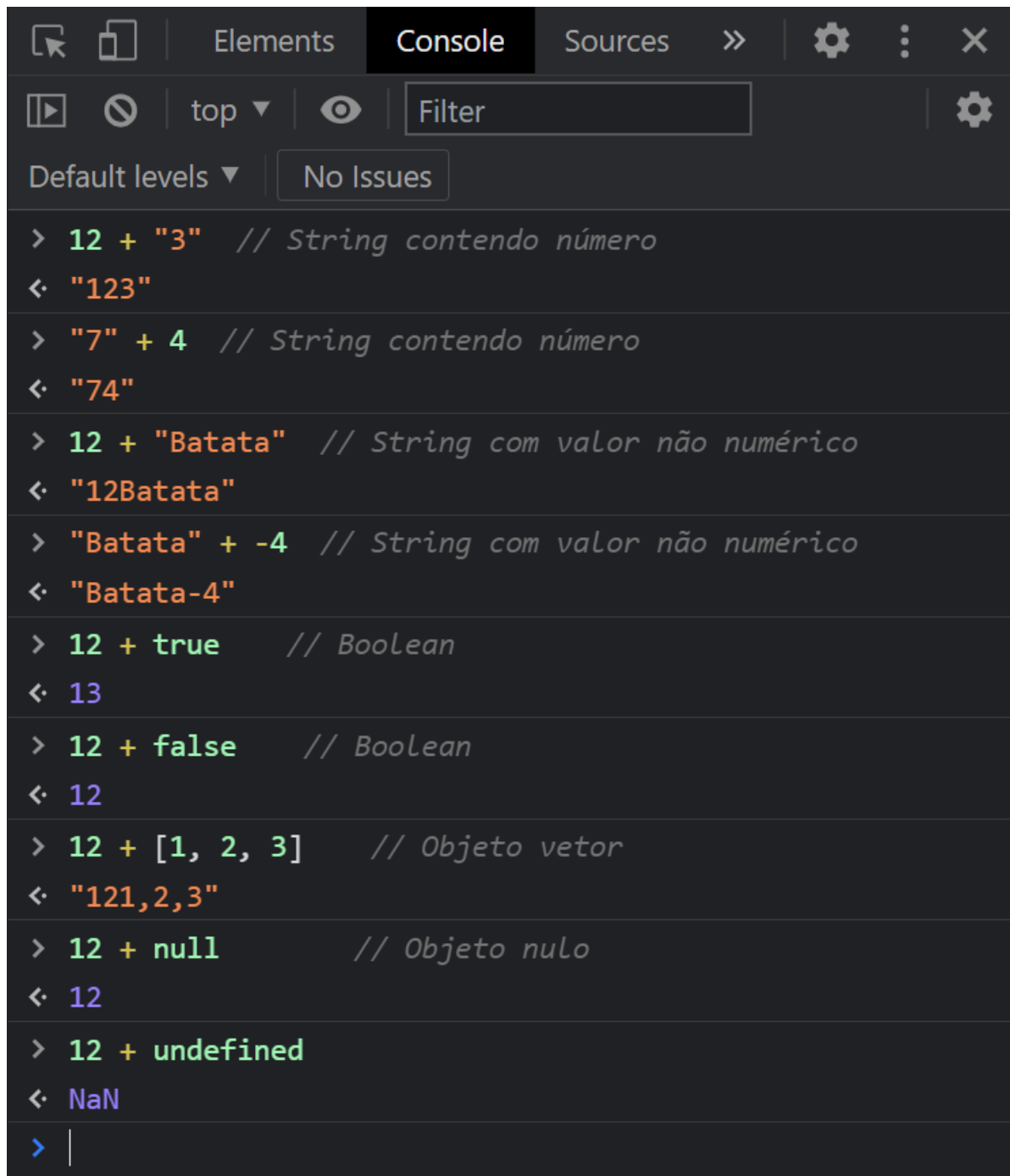
Se você for observador(a), vai notar que eu não mencionei o operador + na observação anterior. Isso foi proposital, pois ele tem um comportamento diferente dos demais (Figura 3.4).

Esquisito? Que nada, vamos entender o porquê desses resultados.

Primeiramente, precisamos aprender que o **operador + tem duas funções** diferentes no JavaScript:

1. Se **todos os operandos forem numéricos**, ou puderem ser convertidos para números (ou tiverem valores equivalentes a números, como **true**, **false** e **null**), ele agirá como um **operador aritmético de adição**, como estamos acostumados.
2. No entanto, se **pelo menos um de seus operandos for uma string**, ele terá a função de **operador de concatenação** de *strings*. Concatenar significa “emendar” uma *string* em outra. O JavaScript irá converter todos os demais operandos em *strings* e concatenará tudo em uma única *string* de resultado.

O operador + em conjunto com algum valor **undefined** sempre retornará NaN, e é um caso à parte.



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays the results of several JavaScript expressions, demonstrating how the '+' operator behaves with non-numeric operands. The expressions and their results are as follows:

- `12 + "3"` (comment: *// String contendo número*) results in `"123"`.
- `"7" + 4` (comment: *// String contendo número*) results in `"74"`.
- `12 + "Batata"` (comment: *// String com valor não numérico*) results in `"12Batata"`.
- `"Batata" + -4` (comment: *// String com valor não numérico*) results in `"Batata-4"`.
- `12 + true` (comment: *// Boolean*) results in `13`.
- `12 + false` (comment: *// Boolean*) results in `12`.
- `12 + [1, 2, 3]` (comment: *// Objeto vetor*) results in `"121,2,3"`.
- `12 + null` (comment: *// Objeto nulo*) results in `12`.
- `12 + undefined` results in `NaN`.

The console interface includes tabs for 'Elements', 'Console', and 'Sources', along with a 'Filter' input field and a 'No Issues' status indicator.

Figura 3.4: O operador '+' com operandos não numéricos

IMPORTANTE

Comprender as duas funções do operador + (quando há e quando não há *strings* entre os operandos) é **FUNDAMENTAL** para evitar frustrações futuras ao tratar com valores informados pelo usuário.

3.1.2 Operadores de atribuição composta

Quem já estudou algoritmos sabe que uma das tarefas mais comuns em programação é a acumulação. Quando estamos somando uma lista de números por exemplo, iniciamos uma variável com o valor 0 e vamos **acumulando** o valor dos números, mais ou menos assim (Listagem 3.1):

Listagem 3.1 Exemplo de acumulação em variável

```
1 // Somando os valores 10, 20, 30 e 40
2 let soma = 0
3 soma = soma + 10
4 soma = soma + 20
5 soma = soma + 30
6 soma = soma + 40
7 // O valor final da variável soma é 100
```

Observe que, no processo de acumulação, a variável `soma` aparece **antes** e **depois** do sinal de atribuição (=). Para casos assim, o JavaScript dispõe de uma série de **operadores de atribuição composta**, como o operador `+=`, que evitam a repetição do nome a variável. Usando-o, o código anterior fica assim (Listagem 3.2):

Listagem 3.2 Acumulação usando o operador '+='

```
1 // Somando os valores 10, 20, 30 e 40
2 let soma = 0
3 soma += 10
4 soma += 20
5 soma += 30
6 soma += 40
7 // O valor final da variável soma é 100
```

Veja alguns dos outros operadores de atribuição composta na Tabela 3.3:

Tabela 3.3: Operadores de atribuição composta mais comuns

Nome	Operador	Exemplo de uso	Significado
Atribuição de subtração	<code>--</code>	<code>x -= 10</code>	<code>x = x - 10</code>
Atribuição de multiplicação	<code>*=</code>	<code>x *= 10</code>	<code>x = x * 10</code>

Nome	Operador	Exemplo de uso	Significado
Atribuição de divisão	/=	x /= 10	x = x / 10
Atribuição de resto de divisão	%=	x %= 10	x = x % 10
Atribuição de potenciação	**=	x **= 10	x = x ** 10

Capítulo 4

ENTRADA E SAÍDA

Capítulo 5

ESTRUTURAS DE CONTROLE

Capítulo 6

ESTRUTURAS DE REPETIÇÃO

Capítulo 7

FUNÇÕES

Capítulo 8

DOCUMENT OBJECT MODEL (DOM)

Capítulo 9

VETORES

Capítulo 10

OBJETOS

Capítulo 11

MANIPULAÇÃO DE *STRINGS*

Capítulo 12

REVISITANDO O DOM

Licença

Esta obra está licenciada a você sob a licença [Creative Commons BY-NC 4.0](#).

Créditos

Imagem da capa: [Technology vector created by vectorjuice - www.freepik.com](#)