MÓDULO IV

Lógica de Programação utilizando a Linguagem JavaScript

A Educação é o primeiro passo para um futuro melhor...

MÓDULO IV

Matemática básica no **JavaScript** — números e operadores

Operador	Nome	Propósito	Exemplo
`+`	Adição	Adiciona um número a outro.	`6 + 9`
`-`	Subtração	Subtrai o número da direita do número da esquerda.	`20 - 15`
`*`	Multiplicação	Multiplica um número pelo outro.	`3 * 7`
`/`	Divisão	Divide o número da esquerda pelo número da direita.	`10 / 5`
`%`	Restante (*Remainder* - as vezes chamado de modulo)	Retorna o resto da divisão em números inteiros do número da esquerda pelo número da direita.	`8 % 3` (retorna 2; como três cabe duas vezes em 8, deixando 2 como resto.)

MÓDULO IV

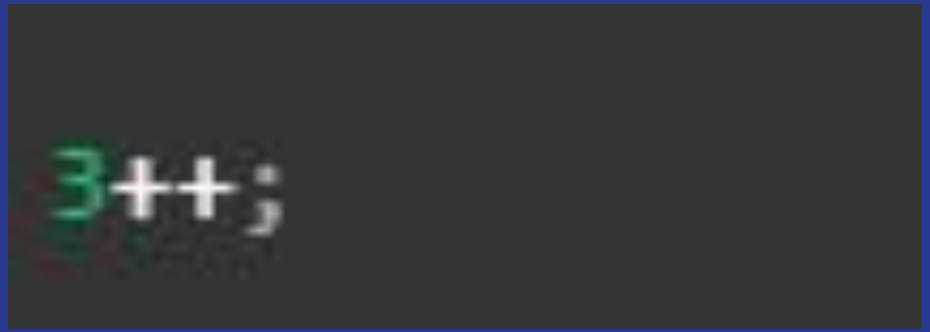
Operadores de incremento e decremento

Às vezes você desejará adicionar ou subtrair, repetidamente, um valor de uma variável numérica. Convenientemente isto pode ser feito usando os operadores incremento **++** e decremento **--**. Usamos **++** em nosso jogo "Adivinhe o número" no primeiro artigo Um primeiro mergulho no **JavaScript**, quando adicionamos **1** ao nosso **contagemPalpites** para saber quantas adivinhações o usuário deixou após cada turno.

```
contagemPalpites++;
```

MÓDULO IV

Vamos tentar brincar com eles no seu console. Para começar, note que você não pode aplicá-las diretamente a um número, o que pode parecer estranho, mas estamos atribuindo a variável um novo valor atualizado, não operando no próprio valor. O seguinte retornará um erro:



MÓDULO IV

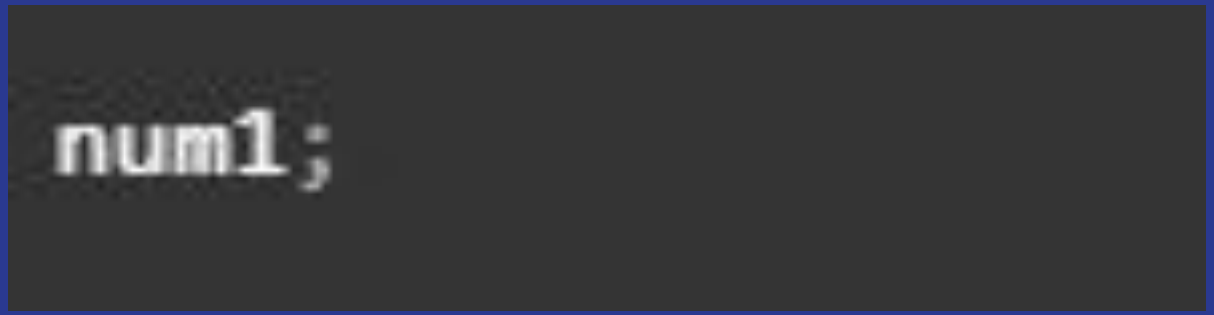
Então, você só pode incrementar uma variável existente. Tente isto:

```
var num1 = 4;

num1++;
```

MÓDULO IV

Ok, segunda coisa estranha! Quando você fizer isso, verá um valor 4 retornado - isso ocorre porque o navegador retorna o valor atual e, *em seguida*, incrementa a variável.
Você pode ver que ele foi incrementado se você retornar o valor da variável novamente:



MÓDULO IV

Acontece a mesma coisa com -- : tente o seguinte

```
var num2 = 6;

num2--;

num2;
```

Você pode fazer o navegador fazer o contrário - incrementar/decrementar a variável e depois retornar o valor, colocando o operador no início da variável ao invés do final. Tente os exemplos acima novamente, mas desta vez use ++num1 e --num2.

MÓDULO IV

Operadores de atribuição

Operadores de atribuição são os que atribuem um valor à uma variável.
Nós já usamos o básico, =, muitas vezes, simplesmente atribuindo à variável do lado esquerdo o valor indicado do lado direito do operador:

```
var x = 3; // x contém o valor 3
var y = 4; // y contém o valor 4
x = y; // x agora contém o mesmo valor de y, 4
```

MÓDULO IV

Mas existem alguns tipos mais complexos, que fornecem atalhos úteis para manter seu código mais puro e mais eficiente. Os mais comuns estão listados ao lado:

Operator	Name	Purpose	Example	Shortcut for
`+=`	Atribuição de adição	Adiciona o valor à direita ao valor da variável à esquerda e, em seguida, retorna o novo valor da variável	`x = 3; x += 4;`	`x = 3; x = x + 4;`
`-=`	Atribuição de subtração	Subtrai o valor à direita do valor da variável à esquerda e retorna o novo valor da variável	`x = 6; x -= 3;`	`x = 6; x = x - 3;`
`*=`	Atribuição de multiplicação	Multiplica o valor da variável à esquerda pelo valor à direita e retorna o novo valor da variável	`x = 2; x *= 3;`	`x = 2; x = x * 3;`
`/=`	Atribuição de divisão	Divide o valor da variável à esquerda pelo valor à direita e retorna o novo valor da variável	`x = 10; x /= 5;`	`x = 10; x = x / 5;`

MÓDULO IV

Tente digitar alguns dos exemplos acima em seu console para ter uma ideia de como eles funcionam. Em cada caso, veja se você pode adivinhar qual é o valor antes de digitar a segunda linha.

Note que você pode muito bem usar outros valores no lado direito de cada expressão, por exemplo:

```
var x = 3; // x contém o valor 3

var y = 4; // y contém o valor 4

x *= y; // x agora contém o valor 12
```

Nota: Existem muitos outros operadores de atribuição disponíveis, mas estes são os básicos que você deve aprender agora.

MÓDULO IV

Entendendo o Escopo de Variáveis em JavaScript: Global, Local e de Bloco

O escopo de uma variável em JavaScript determina onde essa variável é acessível dentro do código. Compreender o escopo é essencial para escrever um código limpo, seguro e funcional, e para isso vamos explorar os diferentes tipos de escopo de variáveis em JavaScript: escopo global, escopo local e escopo de bloco.

MÓDULO IV

Escopo Global:

As variáveis globais são declaradas fora de qualquer função e estão disponíveis em todo o código, dentro de qualquer função ou bloco. Elas podem ser acessadas e modificadas de qualquer lugar do código. No entanto, o uso excessivo de variáveis globais pode levar a problemas de legibilidade, manutenção e possíveis conflitos de nomes.

```
var globalVar = "Variável global (var)";
let globalLet = "Variável global (let)";
const globalConst = "Variável global (const)";
```

MÓDULO IV

Escopo Local:

As variáveis locais são declaradas dentro de uma função e só são acessíveis dentro dessa função, ou seja, são visíveis apenas dentro do bloco em que foram declaradas. Elas são criadas quando a função é chamada e são destruídas quando a função é concluída. Variáveis locais têm prioridade sobre variáveis globais com o mesmo nome dentro da função onde são declaradas.

```
//variáveis globais
var globalVar = "Variável global (var)";
let globalLet = "Variável global (let)";
const globalConst = "Variável global (const)";

//dentro de funções temos variáveis locais
function exemploEscopo() {
var localVar = "Variável local (var)";
let localLet = "Variável local (let)";
}
```

MÓDULO IV

Escopo de Bloco:

O escopo de bloco foi introduzido com o ES6 (ECMAScript 2015) e está associado às variáveis declaradas com **let** e **const** possuindo então escopo restrito ao bloco em que foram declaradas.

Um bloco é delimitado por chaves **{}** e pode ser uma estrutura condicional (**if, else, switch**), um loop (**for, while, do-while**) ou até mesmo um bloco independente.

```
//variáveis globais
var globalVar = "Variável global (var)";
let globalLet = "Variável global (let)";
const globalConst = "Variável global (const)";

//dentro de funções temos variáveis locais
function exemploEscopo() {
var localVar = "Variável local (var)";
let localLet = "Variável local (let)";

//dentro da função agora temos tambem variáveis de bloco
{
  let varBlock = "Variável de bloco (let)";
  const constBlock = "Variável de bloco (const)";
 }
}
```

MÓDULO IV

Como Adicionar JavaScript no HTML

Neste tutorial, vamos mostrar para você como adicionar **JavaScript** no **HTML**. O começo deste guia vai ter uma curta introdução ao **JavaScript**, enquanto o restante do conteúdo vai focar em várias maneiras que ensinam você a como inserir **JavaScript** no **HTML**.

Se você quer exibir no seu site um conteúdo estático como, por exemplo, um conjunto de imagens, então o **HTML** pode servir muito bem para você. Entretanto, páginas estáticas estão cada vez mais se tornando coisa do passado.

A maior parte dos conteúdos de hoje é interativo e possui slideshows chamativos, formulários e menus.

Eles melhoram a experiência do usuário e adicionam dinamicidade para um site, independente do host que você usar.

MÓDULO IV

E isso tudo é conseguido através do uso de linguagens de programação, sendo a **JavaScript** uma das mais famosas nesse sentido. Ela permite que os desenvolvedores criem sites que interajam com os usuários e vice-versa.

Embora existam muitas outras linguagens disponíveis, nenhuma delas é tão popular quanto a **JavaScript**. E para tirar o seu máximo de proveito, ela é geralmente usada em conjunto com o **HTML**.

MÓDULO IV

Vantagens da Linguagem JavaScript

A linguagem **JavaScript** era, antes de tudo, conhecida como **LiveScript**. Por conta do termo **Java** ser muito popular no mundo inteiro, a **Netscape** decidiu renomeá-la para **JavaScript**. A sua origem data de 1995, durante a fase 2.0 da **Netscape**. Abaixo estão algumas das vantagens de usar a linguagem **JavaScript**:

MÓDULO IV

Interação Mínima com o Servidor

É um fato que se você quer otimizar o desempenho de um site, a melhor maneira de fazer isso é reduzindo a comunicação com o servidor.

A **JavaScript** ajuda nesse sentido validando os comandos do usuário pelo cliente.

Ele apenas envia pedidos para o servidor depois de rodar as verificações iniciais de validação.

Como resultado, o uso de recursos e a quantidade de pedidos para o servidor diminuem significantemente.

MÓDULO IV

Interface Rica e Amigável

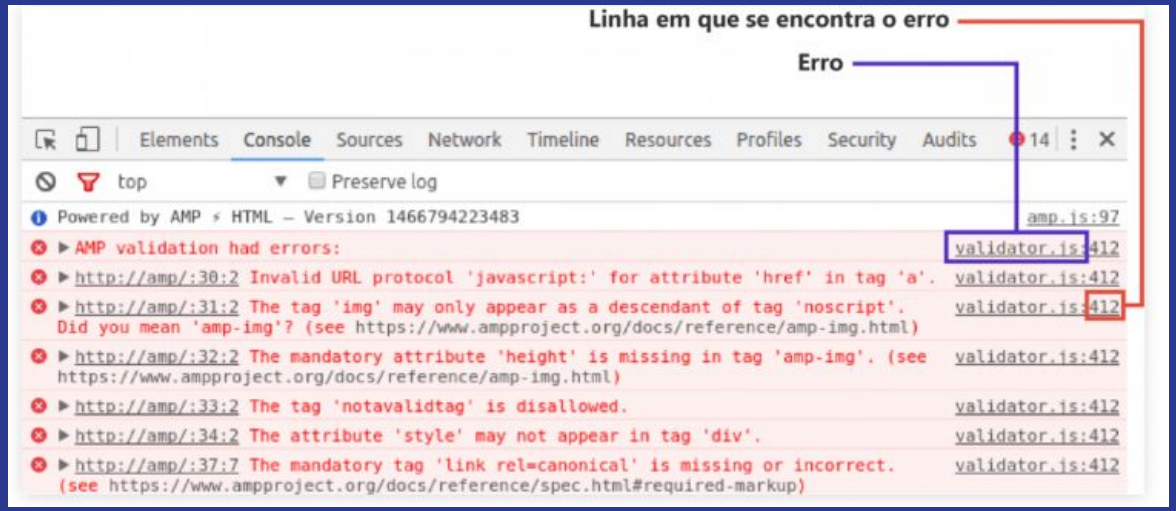
Usando a linguagem JavaScript, você pode criar interfaces interativas pela perspectiva do cliente. Por exemplo: carrosséis nos topos de páginas, slideshows, efeitos que são acionados quando o ponteiro do mouse passa por cima e funções que podem ser arrastadas e movidas de um lugar para outro.

MÓDULO IV

Resposta Imediata ao Usuário

Usando a linguagem **JavaScript**, você assegura respostas imediatas para os usuários. Por exemplo, vamos imaginar uma situação em que um usuário preencheu um formulário e deixou um campo vazio.

Sem a validação **JavaScript**, ele precisa esperar até que a página seja recarregada para só então perceber que deixaram um campo vazio. Entretanto, com a **JavaScript**, ele é alertado sobre isso instantaneamente.



MÓDULO IV

Fácil Depuração

A JavaScript é uma linguagem de interpretação, o que significa que os códigos escritos são decifrados linha a linha. Em caso de surgimento de possíveis erros, você verá o número exato da linha em que o problema se encontra.

MÓDULO IV

Como Adicionar JavaScript no HTML

Existem duas maneiras de adicionar **JavaScript** no **HTML** e fazê-las trabalhar juntas. Agora que já falamos sobre a linguagem **JavaScript** e vimos algumas das vantagens que ela oferece, vamos dar uma olhada em algumas das maneiras que podemos inserir **JavaScript** no **HTML**.

MÓDULO IV

Como Adicionar JavaScript Diretamente em um Arquivo HTML

O primeiro jeito de adicionar JavaScript no HTML é mais direto. Você pode fazer isso usando a tag **<script></script>**, que deve abranger todo código JS que você escrever. O código JS pode ser adicionado:

- Entre as tags **<head>**
- Entre as tags **<body>**

MÓDULO IV

Dependendo de onde você adicionar o código **JavaScript** no arquivo **HTML**, o carregamento é diferente. A prática recomendada é inseri-lo na seção **<head>**, pois ele ficará separado do conteúdo real do arquivo **HTML**.

Mas saiba que, colocá-lo na seção <**body**> poderá melhorar a velocidade de carregamento, já que o conteúdo do site atual será carregado mais rapidamente, e somente então o **JavaScript** será analisado. Para este exemplo, vamos dar uma olhada no arquivo **HTML** abaixo, usado para mostrar o horário atual.

MÓDULO IV

```
<!DOCTYPE html>
<html lang="en-US">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<script>JAVASCRIPT IS USUALLY PLACED HERE</script>
<title>Time right now is: </title>
</head>
<body>
<script>JAVASCRIPT CAN ALSO GO HERE</script>
</body>
</html>
```

Neste formato, o código acima não tem **JavaScript** e, consequentemente, não pode mostrar a hora atual. Podemos adicionar o próximo código para ter certeza de que a página irá mostrar a hora esperada correta.

MÓDULO IV

Vamos colocar este código entre as tags **<script>** e **</script>** e colocá-lo no topo do código **HTML** para garantir que, quando a página carregar, um alerta é gerado e mostre a hora atual para o usuário.

Abaixo você vê como o arquivo **HTML** deve ficar depois de inserir o código JavaScript nele:

```
var time = new Date();
console.log(time.getHours() + ":" + time.getMinutes() + ":" +
time.getSeconds());
```

MÓDULO IV

```
<!DOCTYPE html>
<html lang="en-US">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Time right now is: </title>
<script>
var time = new Date();
console.log(time.getHours() + ":" + time.getMinutes() + ":" +
time.getSeconds());
</script>
</head>
<body>
</body>
</html>
```

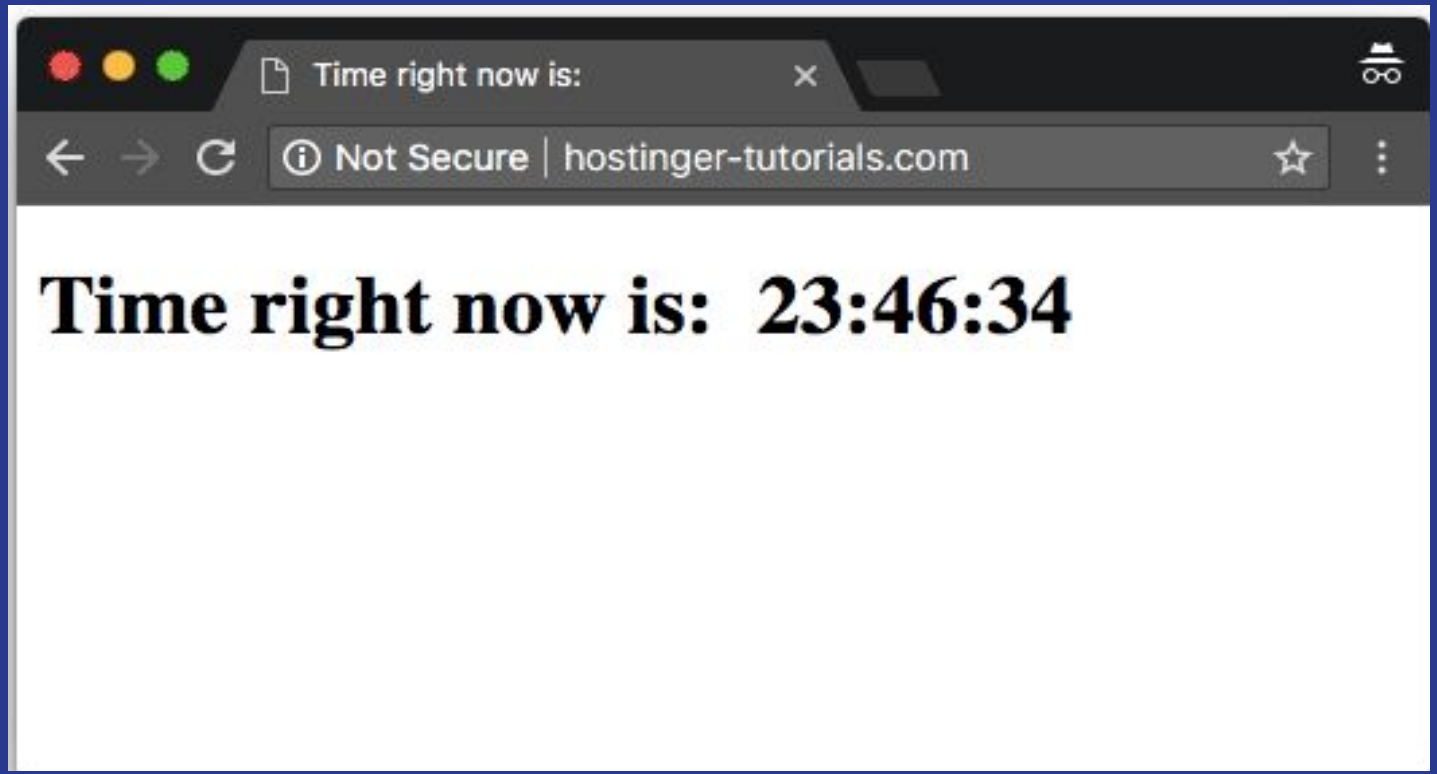
MÓDULO IV

Se você quiser mostrar a hora dentro do corpo da página, você precisa incluir o script dentro das tags <body> do HTML da página. Abaixo você confere como o código deve ficar quando você fizer isso:

```
<!DOCTYPE html>
<html lang="en-US">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Time right now is: </title>
</head>
<body>
<script>
let d = new Date();
document.body.innerHTML = "<h1>Time right now is:  " + d.getHours() + ":" +
d.getMinutes() + ":" + d.getSeconds()
"</h1>"
</script>
</body>
</html>
```

MÓDULO IV

E abaixo está o resultado final do que você deve ver na página:



MÓDULO IV

Como Adicionar JavaScript em um Arquivo Separado

Às vezes, inserir **JavaScript** no **HTML** de forma direta pode não parecer o melhor jeito de ligar uma linguagem à outra. Isso porque porque alguns scripts **JS** precisam ser usados em múltiplas páginas e, por isso, é melhor manter o código **JavaScript** em arquivos separados.

Por isso que um jeito mais aceitável de inserir **JavaScript** no **HTML** é pela importação externa de um arquivo. Esses arquivos podem ser referenciados de dentro de documentos **HTML**, assim como referenciamos documentos **CSS**.

MÓDULO IV

Alguns dos benefícios de adicionar um código JS em arquivos separados são:

- Quando o código **HTML** e o código **JavaScript** são separados, o princípio de design da separação de interesses é cumprido e torna tudo muito mais sustentável e reutilizável.
- A leitura e a manutenção do código ficam muito mais fáceis de se fazer.
- Arquivos **JavaScript** em cache melhoram o desempenho geral de um site ao diminuir o tempo que as páginas carregam.

MÓDULO IV

Podemos referenciar um arquivo **JavaScript** dentro do **HTML** dessa forma abaixo:

```
<!DOCTYPE html>
<html lang="en-US">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Time right now:</title>
</head>
<body>
</body>
<script src="js/myscript.js"></script>
</html>
```

MÓDULO IV

Os conteúdos do arquivo myscript.js serão:

```
et d = new Date();
document.body.innerHTML = "<h1>Time right now is:  " + d.getHours() + ":" +
d.getMinutes() + ":" + d.getSeconds()</h1>"
```

Importante! Aqui fica implícito que o arquivo **myscript.js** está presente no **mesmo diretório** que o arquivo **HTML**.

MÓDULO IV

Exemplo de JavaScript para Validar um Endereço de E-mail

A linguagem **JavaScript** capacita a sua aplicação ajudando você a validar a entrada do usuário pelo cliente. Uma das entradas mais importantes feitas pelos usuários e que frequentemente precisam de validação é o endereço de *e-mail*.

MÓDULO IV

A função **JavaScript** abaixo pode ajudar você a validar um endereço de e-mail antes de enviá-lo para um servidor:

```
function validateEmailAddress(email) {
var re = /^(([^<>()[\]\\.,;:\s@\"]+(\.[^<>()[\]\\.,;:\s@\"]+)*)|(\".+\"))@((
[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]
{2,}))$/;
return re.test(email);
}
function validate() {
$("#result").text("");
var emailaddress = $("#email").val();
if (validateEmailAddress(emailaddress)) {
$("#result").text(emailaddress + " is valid :)");
$("#result").css("color", "green");
} else {
$("#result").text(emailaddress + " is not correct, please retry:(");
$("#result").css("color", "red");
}
return false;
}
$("#validate").bind("click", validate);
```

MÓDULO IV

Para anexar essa função para uma entrada de formulário, você pode usar o código abaixo:

```
<form>
<p>Enter an email address:</p>
<input id='email'>
<button type='submit' id='validate'>Validate!</button>
</form>
<h2 id='result'></h2>
```

MÓDULO IV


Not Secure | hostinger-tutorials.com
Enter an email address:
test@hostinger.com
Validate!
test@hostinger.com is valid :)

MÓDULO IV

Abaixo está o resultado que você obtém depois de combinar todos os ingredientes juntos em um arquivo HTML.

E se a validação for incorreta, o resultado será diferente:

MÓDULO IV

Convertendo tipos de dados em JavaScript

Às vezes precisamos informar em nosso código exatamente o tipo de dados que deve ser usado no processamento de informações.

Como o JavaScript é uma linguagem não-tipada, existe o risco de um tipo de dado ser interpretado de forma errônea em alguns casos particulares, causando problemas no processamento dos dados.

Podemos converter entre tipos de dados usando funções específicas no **JavaScript**, como os métodos **parseInt()** e **parseFloat()**, entre outras.

MÓDULO IV

Operador typeof

Como operador **typeof()** podemos descobrir o tipo de um dado ou de uma variável. Veja os exemplos a seguir:

```
<script>
  document.write(typeof "Fábio" + "<br />");
  document.write(typeof 345.89 + "<br />");
  document.write(typeof 12 + "<br />");
  document.write(typeof true + "<br />");
</script>
```

MÓDULO IV

Métodos Numéricos

Método	Descrição
Number()	Retorna um Number, convertido do seu argumento
parseFloat()	Analisa uma string e retorna um número de ponto flutuante
parseInt()	Analisa uma string e retorna um número inteiro

MÓDULO IV

Conversão de Números em Strings

O método **toString()** é o método interno de um objeto **Number** que permite a mudança de qualquer valor do tipo **Number** para um tipo **String**.

Para utilizar é fácil, basta chamar o método em um valor do tipo Number.

```
let num = 15;
var str = num.toString();
console.log(num); // 15
console.log(str); // "15"
```

MÓDULO IV

Sistema que são válidos para a conversão:

- Sistema **binário** (de base 2), que têm dois algarismos – 0 e 1
- Sistema **ternário** (de base 3), que têm três algarismos – 0, 1 e 2
- Sistema **quaternário** (de base 4), que têm quatro algarismos – 0, 1, 2 e 3
- E assim por diante até o sistema **Hexatridecimal** (de base 36), que tem a combinação dos números arábicos (de 0 a 9) com as letras latinas de A até Z

*Utilizando o método **toString()**, podemos fazer também a conversão de um número decimal para obter um valor em binário.*

MÓDULO IV

Da *seguinte maneira:*

```
var str = (5).toString(2);
console.log(str); // "101"
```

Assim como podemos fazer a utilização de parâmetros radix:

```
Number.toString(radix);
```

MÓDULO IV

O parâmetro “**radix**” refere-se à base númerica utilizada na interpretação de uma string como um número inteiro.

Por padrão, o método **toString()** converte um número para sua representação em base 10. No entanto, você pode usar o parâmetro para converter o número em uma base diferente.

```
let numero = 10;
console.log(numero.toString()); // Saída: "10" (em base 10)
console.log(numero.toString(2)); // Saída: "1010" (em base 2, binária)
console.log(numero.toString(8)); // Saída: "12" (em base 8, octal)
console.log(numero.toString(16)); // Saída: "a" (em base 16, hexadecimal)
```

MÓDULO IV

Também podemos converter uma Array para uma String utilizando o método:

```
let arr = [ "Tok", "Digital" ];
let str = arr.toString();

console.log(str); // "Tok,Digital"
```

Tal como booleanos:

```
let bool = true;
let str = bool.toString();

console.log(str); // "true"
```

MÓDULO IV

O Operador Unary

Unário Mais +
que converte um operando em um número.

```
let a = 10;
a = +a; // 10
```

Unário Menos -
que converte um operando em um número e nega o valor depois dele.

```
let a = 10;
a = -a; // -10
```

MÓDULO IV

Se o valor é um número, o operador unário ***plus*** não causa efeito algum, no entanto o operador unário ***minus*** torna o valor **negativo**.

Caso você aplique os operadores unários ***plus*** ou o ***minus*** em um valor não numérico, ele realiza a mesma conversão que a função **Number()**.

```
let s = "10";

console.log(+s); // 10
```

MÓDULO IV

Nesse exemplo, **s** é uma ***string***. Entretanto, quando colocamos o operador unário ***plus*** na frente dele, a ***string*** s é convertida em um **número**.

A seguir podemos ver que o operador unário converte valores **booleanos** em números, **false** em **0** e **true** em **1**.

```
let verdadeiro = true;
let falso = false;
console.log(+verdadeiro); // 1
console.log(+falso); // 0
```

MÓDULO IV

Operadores de incremento e decremento

Inspirado na linguagem de programação C, o conceito de incremento possui um significado sugestivo: aumentar um valor existente em **1** unidade. Da mesma forma, o decremento realiza o oposto, diminuindo o valor em **1** unidade. Esses operadores podem ser aplicados como **prefixos** ou **sufixos**, e sua escolha afeta o momento em que a operação é realizada em relação à avaliação da expressão

MÓDULO IV

Incremento prefixo e sufixo ++
que adiciona um ao seu operando.

```
let inc = 9;
++inc;
console.log(inc); // 10
```

No exemplo acima o operador de **incremento** como prefixo adiciona sobre o valor de **inc**, **1** unidade a mais.

MÓDULO IV

Decremento prefixo e sufixo --
que subtrai um do seu operando.

```
let dec = 10;
--dec;
console.log(dec); // 9
```

MÓDULO IV

O **incremento** e **decremento** agora atuando como sufixo funciona da mesma maneira que utilizado como prefixo. A única diferença é que ao invés da operação ser executada depois da resolução, utilizando o sufixo, a operação é executada antes mesmo da resolução.

Prefixo:

```
let i = 10, j = 20;
let op = i-- + j;
console.log(op); // 30
console.log(op); // 9
```

MÓDULO IV

Sufixo:

```
i = 10;
let op = --i + j;
console.log(op); // 29
console.log(i); // 9
```

MÓDULO IV

Quando você usa o operador de incremento ou decremento como sufixo (**i--** ou **i++**), a operação é realizada após a resolução da expressão.
Isso significa que, no caso de **i--**, o valor de **i** é usado na expressão, e então **i** é **decrementado**.

```
console.log(op); // 30
console.log(op); // 9
```

MÓDULO IV

Por outro lado, quando você usa o operador como prefixo (**--i** ou **++i**), a operação é realizada antes da resolução da expressão. Portanto, no caso de **--i**, **i** é primeiro **decrementado** e, em seguida, o valor de **i decrementado** é usado na expressão.

```
console.log(op); // 29
console.log(i); // 9
```

MÓDULO IV

Tabela de Conversão de Tipos JavaScript

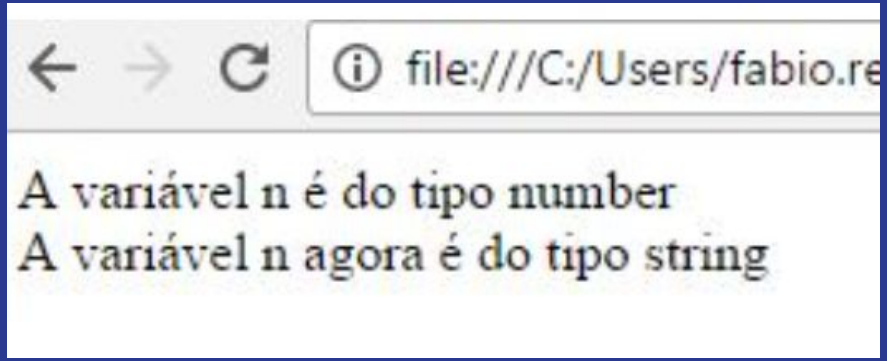
De/Para	String	Number	Boolean	Object	Undefined	Null
String	–	String()	Boolean()	Object()	–	String()
Number	Number()	–	Boolean()	Object()		
Boolean	String()	Number()	–	Object()	–	–
Object	toString()	Number()	Boolean()	–	–	–
Undefined	String()	NaN	false	TypeError	–	–
Null	String()	0	false	Object()	–	–

MÓDULO IV

Converter números para strings

Para converter um **número** em **string**, use o método **String()**, como no exemplo a seguir:

```
<script>
  n = 20;
  document.write("A variável n é do tipo " + typeof n + "<br
/>");
  n = String(n);
  document.write("A variável n agora é do tipo " + typeof n +
"<br />");
</script>
```



Podemos também usar o método **toString()** do objeto Number para realizar essa conversão:

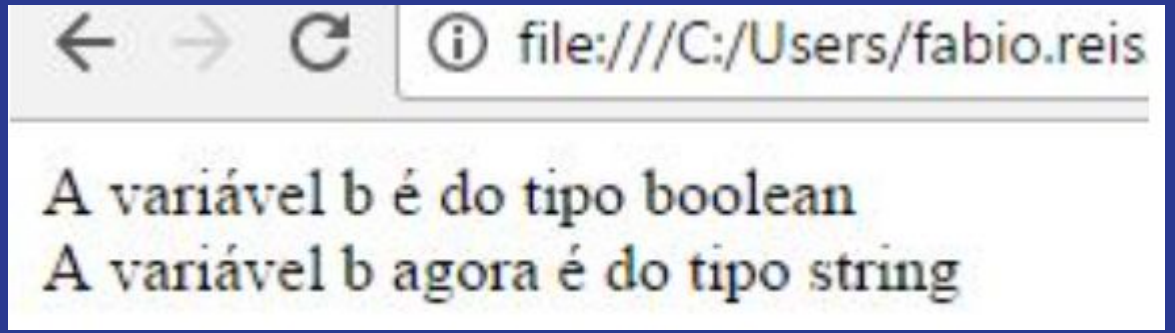
```
n = n.toString();
```

MÓDULO IV

Convertendo valores lógicos para strings

O método global **String()** também pode converter um valor lógico (**booleano**) para **string**:

```
<script>
b = true;
document.write("A variável b é do tipo " + typeof b + "<br />");
b = String(b);
document.write("A variável b agora é do tipo " + typeof b + "<br
/>");
</script>
```



MÓDULO IV

Podemos também usar o método **toString()** do objeto **Boolean**:

```
b = b.toString();
```

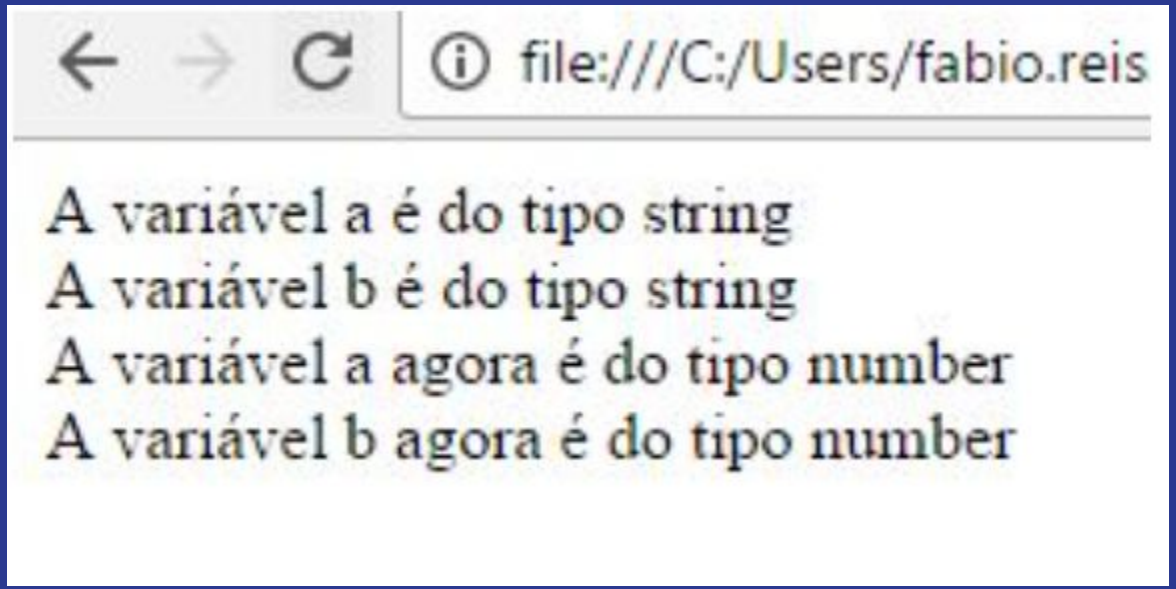
MÓDULO IV

Convertendo uma string em número

Para converter uma string em um número podemos usar o método Number():

```
<script>
 a = "10";
 b = "23.6";
 document.write("A variável a é do tipo " + typeof a + "<br />");
 document.write("A variável b é do tipo " + typeof b + "<br />");
 a = Number(a);
 b = Number(b);
 document.write("A variável a agora é do tipo " + typeof a + "<br
/>");
 document.write("A variável b agora é do tipo " + typeof b + "<br
/>");
</script>
```

MÓDULO IV


file:///C:/Users/fabio.reis
A variável a é do tipo string
A variável b é do tipo string
A variável a agora é do tipo number
A variável b agora é do tipo number

MÓDULO IV

Convertendo **string** para **número inteiro**

Podemos converter uma string especificamente para um inteiro usando o método **parseInt()**:

```
<script>
 a = "10";
 document.write("A variável a é do tipo " + typeof a + "<br />");
 a = parseInt(a);
 document.write("A variável a agora é do tipo " + typeof a + "<br
/>");
 document.write("Valor de a: " + a +  "<br />");
 b = "12.45";
 document.write("A variável b é do tipo " + typeof b + "<br />");
 document.write("Valor de b: " + b + "<br />");
 b = parseInt(b);
 document.write("A variável b agora é do tipo " + typeof b + "<br
/>");
 document.write("Valor de a: " + a +  "<br />");
 document.write("Valor de b: " + b);
</script>
```

MÓDULO IV

Convertendo string para número de ponto flutuante (**float**)

Podemos converter uma **string** especificamente para um número de ponto flutuante usando o método **parseFloat()**:

```
<script>
 a = "10";
 document.write("A variável a é do tipo " + typeof a + "<br />");
 a = parseFloat(a);
 document.write("A variável a agora é do tipo " + typeof a + "<br
/>");
</script>
```

MÓDULO IV

Precisão de ponto flutuante

Quando trabalhamos com números de ponto flutuante. podemos controlar a precisão que será retornada (quantidade de casas decimais exibidas) usando o método **toFixed()**, o qual retorna uma **string**:

```
<script>
 a = 10.2;
 document.write("O valor de a é " + a.toFixed(2));
</script>
```

MÓDULO IV

Os valores são arredondados com o método **toFixed()** se o número de casas a exibir for menor que o número de casas decimais que o número possui.

Tanto **toFixed()** quanto **toPrecision()** (próxima seção) são métodos do objeto Number.

MÓDULO IV

EXERCÍCIOS...

A Educação é o primeiro passo para um futuro melhor...

MÓDULO IV

FIM...

A Educação é o primeiro passo para um futuro melhor...