MÓDULO I

Lógica de Programação utilizando a Linguagem JavaScript

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

MÓDULO I

Ambientes de Desenvolvimento: Configuração para Programação em JavaScript

JavaScript Dentro dos Navegadores

O fato de navegadores para Internet permitirem o acesso a um interpretador embutido em computadores contribui por tornar JavaScript uma das linguagens mais simples para iniciar atividades de programação. Podendo simplesmente, abrir as ferramentas para desenvolvedores e escrever o código-fonte no interpretador mediado pelo console.

Caso ainda não tenha um, o primeiro passo é escolher e instalar um (*ou mais)* navegador(es) de Internet moderno(s), como:

- **Mozilla Firefox;**
- **Google Chrome**
- **Microsoft Edge**
- **Apple Safari;**
- **Opera.**

MÓDULO I

Desenvolvedoras e desenvolvedores Web costumam instalar vários navegadores, para que possam testar suas páginas e sistemas neles. Infelizmente, existem diferenças entre navegadores de Internet que podem resultar em implementações que funcionam em alguns, mas não funcionam em outro. Simplificadamente, existem três principais motores para navegadores, com os quais se pode criar um navegador de Internet:

1. **Gecko**, usando pelo **Firefox**;
2. **Webkit**, usado pelo **Safari**;
3. **Blink**, usado pelo **Chrome**.

Existem outros motores (como **KHTML**) e inúmeros outros navegadores. Contudo, muitos dos outros navegadores utilizam um dos três motores principais.
Não é necessário instalar todos os navegadores possíveis para iniciar atividades de programação.
Um bom ponto de partida é instalar seu navegador favorito, ou instalar um navegador que utilize cada um dos três motores principais.

MÓDULO I

Acessibilidade para a Internet

Contudo, existe uma exceção. Acessibilidade é importante e nem todas as pessoas usam a Internet de forma gráfica. Logo, convém conhecer navegadores textuais como:

- Lynx;
- Links.

Pessoas com deficiência visual, inclusive pessoas cegas, também usam a Internet. Contudo, elas só podem acessar todo o conteúdo caso a página seja concebida para isso. Para tornar suas páginas e sistemas acessíveis para mais pessoas, consulte referências como as Diretrizes de Acessibilidade para Conteúdo Web (**WCAG**) 2.0.

MÓDULO I

JavaScript Fora dos Navegadores

Para começar, o interpretador do navegador é mais prático e conveniente, por não requerer a instalação e configuração dos interpretadores. Contudo, caso não queira usar o interpretador de um navegador, outra possibilidade é instalar um interpretador **JavaScript** para linha de comando. Duas opções populares são:

1. SpiderMonkey;
2. Node.js.

A possibilidade de automatizar execução e teste de código-fonte são duas vantagens de optar-se pelo de **SpiderMonkey** ou **Node.js**. Em particular, existem gerenciador de pacotes (como **npm**) que facilitam a obtenção de bibliotecas para programação em **JavaScript**

MÓDULO I

Além disso, interpretadores externos permitem usar JavaScript em aplicações que não sejam feitas para a Internet. A programação Web é comumente dividida em duas frentes:

1. **Front-end**, que é a parte com a qual usuários finais interagem. O ***front-end*** é a parte visível e audível de uma página da Internet. Além de **JavaScript**, linguagens como Linguagem de Marcação de Hipertexto (***HyperText Markup Language* ou HTML**) e Folhas de Estilo em Cascata (***Cascading Style Sheets*** ou **CSS**) são comuns para programação em ***front-end***. Após concepção, **JavaScript** rapidamente tornou-se uma linguagem ***front-end*** de páginas Web (*embora não se utilizasse o termo na época*) com sua inclusão no extinto navegador *Netscape Navigator*;
2. ***Back-end***, que é a parte com a qual usuários finais não interagem (*ao menos diretamente*), mas que permite que sistemas Web complexos funcionem. A rigor, pode-se escrever código para ***back-end*** em qualquer linguagem de programação. Linguagens mais comuns incluem **PHP, Java, C#, Ruby, Python, Go, C, C++ e... JavaScript**.

MÓDULO I

Os méritos de **JavaScript** como linguagem de programação para ***back-end*** em uso profissional podem ser discutíveis. Contudo, ela é usada e os altos investimentos resultaram em melhorias para a linguagem como um todo, sobretudo em ferramentas de suporte. Projetos como **Babel** e **Unified** são alguns exemplos.

Na prática, o resultado é positivo para iniciantes. A possibilidade de usar linguagem fora de navegadores, a proliferação de sistemas Web dinâmicos e complexos, e de bibliotecas para programação podem facilitar a transição de nível iniciante para intermediário e avançado.

Além disso, iniciantes comumente querem visualizar resultados fora de um terminal. Criar conteúdo multimídia *(por exemplo, com imagens, sons e vídeos*) em **JavaScript** para sistemas Web é muito mais simples que em muitas outras linguagens de programação. Com **HTML, CSS** e **JavaScript**, isso é trivialmente simples se comparado a outras linguagens tradicionais.
Para programação como carreira, **JavaScript** tanto para ***front-end***, quando para ***back-end*** ou mesmo ambos (*combinação chamada de full stack*) é uma opção válida e com boa empregabilidade. A demanda por profissionais que programem para Web é alta e tende a continuar a crescer com a digitalização de negócios e outras atividades.

MÓDULO I

Primeiros Passos para Começar a Programar em JavaScript

Existem duas formas de começar rapidamente a programar em **JavaScript**.

1. Como mencionado, pode-se simplesmente, abrir as ferramentas para desenvolvedores em um computador e escrever o código-fonte no interpretador mediado pelo console.
2. Pode-se criar dois arquivos de texto (um arquivo **HTML** e um arquivo **JavaScript**) e abrir o arquivo **HTML** no navegador. Opcionalmente, pode-se criar um terceiro arquivo contendo definições para estilo em **CSS** para personalizar a aparência de sua página.

MÓDULO I

Primeiros Passos com o Console Embutido

Em muitos navegadores para computadores, pode-se apertar **F12** para iniciar o console embutido no navegador. Comece agora mesmo:

A. Aperte F12;
B. Procure a aba Console, Terminal, Interpreter ou algum termo semelhante no painel que aparecer;
C. Digite algum código. Por exemplo:

```
console.log("Olá! Meu nome é Igor. Sou um programador.")
```

1. A documentação para **console.log()** está disponível disponível na **MDN**. De acordo com a documentação **console.log()** escreve uma mensagem no console.
2. Aperte **enter** para enviar o código para o interpretador;
3. Aguarde a resposta.
   O resultado será:

```
Olá! Meu nome é Igor. Sou um programador.
```

MÓDULO I

Conhecendo JavaScript por Meio de Experimentação

Prática contínua é fundamental em programação. Os próximos blocos de código ilustram alguns recursos da linguagem **JavaScript** (*e que também existem em muitas outras linguagens de programação*).

Neste momento, não se preocupe caso você não entenda completamente o que cada um deles faz. Apenas experimente. Escreva os blocos de código no interpretador fornecido pelo console (*reabra com **F12**, caso necessário*), execute o código (*aperte **enter***) e verifique os resultados. Se quiser, altere valores para observar novos resultados. Caso a alteração resulte em erro, compare com o original e note as diferenças.

Além disso, muitos itens estão acompanhados de entrada de documentação usando a **MDN**.
Caso queira saber como um recurso da linguagem funcione, você pode consultar a referência.
Lembre-se, contudo, que a intenção é mostrar que não é preciso decorar palavras e expressões para programar. Você pode consultá-las quando precisar. Logo, a intenção não é assustar você com mais material ou mostrar tudo que se deve saber. Pelo contrário; o entendimento virá com tempo e prática.

MÓDULO I

Experimente, observe os resultados e faça alterações. Comece a aprender de forma lúdica e interativa.

```
console.log("Olá! Meu nome é Igor. Sou um programador.")
Olá! Meu nome é Igor. Sou um programador.
undefined
console.log("Uma linha.")
Uma linha.
undefined
console.log("Outra linha.")
Outra linha.
undefined
console.log("Uma linha.\nOutra linha.")
Uma linha.
Outra linha.
undefined
console.log("Interpretador processa") // Interpretador ignora.
Interpretador processa
undefined
/* <- Aqui começa o comentário.
Ele pode ter várias linhas.
Aqui ele termina -> */
undefined
/* Ele também pode começar e terminar na mesma linha. */
undefined
// Contudo, para isso, é mais fácil usar este estilo de comentário.
```

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

MÓDULO I

Expressões matemáticas:

```
console.log(1 + 2 - 3 + (4 * 5) / 6)
3.3333333333333335
undefined
```

Cálculos matemáticos:

```
console.log(1 + 1)
console.log(2 - 2)
console.log(3 * 3)
console.log(4 / 4)
2
0
9
1
```

Potenciação

```
console.log(5 ** 2)
25
```

Raiz quadrada:

```
console.log(Math.sqrt(25))
5
undefined
```

Operações trigonométricas, como seno :

```
console.log(Math.sin(Math.PI / 2.0)) // Seno.
1
undefined
```

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

MÓDULO I

Comparações (*true significa: verdadeiro; false: significa falso*):

Outras comparações:

```
console.log(1 < 2) // Menor.
console.log(1 > 2) // Maior.
console.log(1 <= 2) // Menor ou igual.
console.log(1 >= 2) // Maior ou igual.
```

```
true
false
true
false
undefined
```

Igualdade e uniformidade:

```
console.log(1 === 2) // Igual: os três iguais são necessários!
console.log(1 !== 2)
```

```
false
true
```

```
console.log("Igor" === "Igor")
console.log("Igor" !== "Igor")
console.log("Igor" !== "Seu Nome")
// JavaScript considera que letras minúsculas são diferentes de maiúsculas (e vice-versa).
console.log("I" === "i")
console.log("I" !== "i")
console.log("Igor" === "igor")
console.log("Igor" !== "igor")
```

```
true
false
true
false
true
false
true
```

MÓDULO I

Variáveis e atribuição :

Variáveis são como caixinhas que armazenam valores colocados dentro delas.
Efetiva-se um armazenamento usando o operador de atribuição, que é um único igual **(=)** em **JavaScript**.

```
let x = 123
console.log(x)
```

```
123
```

```
undefined
```

```
let resultado = 123.456 + 987.654
console.log(resultado)
```

```
1111.11
```

```
undefined
```

MÓDULO I

Deve-se notar que a declaração de variáveis pode ser feita uma única vez por nome escolhido. Entretanto, pode-se mudar seu valor quantas vezes forem necessárias.

```
let nome = "Igor"
console.log(nome)
Igor
undefined
nome = "Igor M. L."
console.log(nome)
Igor M. L.
undefined
```

```
let variaveis_podem_variar = "Igor"
console.log(variaveis_podem_variar)
Igor
undefined
variaveis_podem_variar = "João"
console.log(variaveis_podem_variar)
João
undefined
variaveis_podem_variar = "Maria"
console.log(variaveis_podem_variar)
Maria
undefined
```

```
let valor_logico = true
console.log(valor_logico)
true
undefined
valor_logico = false
console.log(valor_logico)
false
undefined
valor_logico = (1 + 1 === 2)
true
```

MÓDULO I

Constantes

```
const PI = 3.14159
console.log(PI)
const E = 2.71828
console.log(E)
```

```
3.14159
```

```
2.71828
```

```
undefined
```

```
E = 0 // Erro; o valor de uma constante é inalterável após definido.
```

```
▶Uncaught TypeError: Assignment to constant variable.
    at <anonymous>:1:3
```

MÓDULO I

Erros:

```
console.log(Ooops!) // Texto deve estar entre aspas duplas, simples, ou acentos graves.
Uncaught SyntaxError: missing ) after argument list
cs1.log("Nome incorreto para console")
▶ Uncaught ReferenceError: cs1 is not defined
    at <anonymous>:1:1
console.log(1 / 0)
Infinity
undefined
console.log(0 / 0)
NaN
undefined
```

MÓDULO I

Cadeias de caracteres (*strings*) para palavras e texto :

```
console.log("Ok, cadeia de caracteres válida")
Ok, cadeia de caracteres válida
undefined
console.log('Ok, outra cadeia de caracteres válida')
Ok, outra cadeia de caracteres válida
undefined
console.log(`Ok, também é uma cadeia de caracteres válida`)
Ok, também é uma cadeia de caracteres válida
undefined
```

```
console.log("Para 'misturar' `aspas`, use diferentes das externas.")
Para 'misturar' `aspas`, use diferentes das externas.
undefined
console.log("Caso contrário, você precisa escapá-las com uma contrabarra, assim: \". A contrabarra é necessária.")
Caso contrário, você precisa escapá-las com uma contrabarra, assim: ". A contrabarra é necessária.
undefined
```

MÓDULO I

Operações lógicas :

```
console.log(true && true) // Operação lógica "e" ("and").
true
undefined
console.log(true || false) // Operação lógica "ou" ("or").
true
undefined
console.log(!true) // Operação lógica "não" ("not").
false
undefined
```

MÓDULO I

Condições

```
let eu_uso = "Brave"
eu_uso = eu_uso.toLowerCase()
if (eu_uso === "firefox") {
 console.log("Você usa um navegador da Mozilla.")
} else if ((eu_uso === "chrome") || (eu_uso === "chromium")) {
 console.log("Você usa um navegador do Google.")
} else if (eu_uso === "edge") {
 console.log("Você usa um navegador da Microsoft.")
} else if (eu_uso === "safari") {
 console.log("Você usa um navegador da Apple.")
} else if (eu_uso === "internet explorer") {
 console.log("Você deveria usar um navegador mais moderno...")
} else {
 console.log("Você usa outro navegador.")
}
```

```
Você usa outro navegador.
```

```
let navegador = "Firefox"
if (navegador === "Firefox") {
 console.log("Mozilla Firefox.")
}
```

```
Mozilla Firefox.
```

```
let meu_navegador = "Brave"
if (meu_navegador === "Firefox") {
 console.log("Você usa um navegador da Mozilla.")
} else {
 console.log("Você usa outro navegador.")
}
```

```
Você usa outro navegador.
```

MÓDULO I

Repetições ou laços

```
for (let i = 0; i < 5; i = i + 1) {
 console.log(i)
}
```

```
0
1
2
3
4
undefined
```

```
let j = 0
while (j < 5) {
 console.log(j)
 ++j // O mesmo que j = j + 1 e similar a j++ (existem algumas pequenas diferenças)
}
```

```
0
1
2
3
4
5
```

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

```
0
1
2
3
4
4
```

```
function minha_funcao(x, y) {
    let resultado = x + y
    return resultado
}
// Uma função é um bloco de código que realiza um processamento arbitrário
// conforme definido pela programadora ou pelo programador.
// Após a definição, pode-se executar a função quando se quiser, por meio
// de uma chamada de função.
```

```
undefined
```

```
z = minha_funcao(12, -34) // Este é um exemplo de uma chamada de função.
console.log(z)
```

```
-22
undefined
```

```
console.log(minha_funcao(1, 2)) // Este é outro.
```

```
3
```

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

MÓDULO I

Tipos de dados :

```
let numero_inteiro = 1
let outro_numero_inteiro = -1
let numero_real = 1.23
let valor_logico = true // ou false; só pode ser true ou false.
let cadeia_caracteres = "Texto aqui. Quebras de linha usam\nou seja, isto estará\n na terceira linha."
```

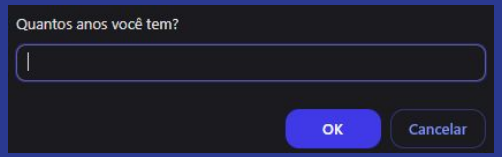
MÓDULO I

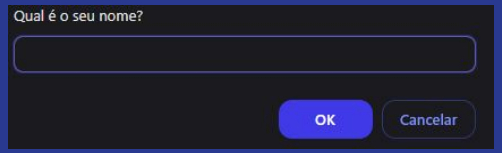
Entrada: (**prompt()**; **parseInt()**):

```
// Quando se solicitar a entrada de um valor, escreva um e aperte `enter`.
let seu_nome = prompt("Qual é o seu nome?")
// parseInt() converte número em texto para número inteiro.
let sua_idade = parseInt(prompt("Quantos anos você tem?"))
console.log(seu_nome)
console.log(sua_idade)
```

```
Igor
```

```
30
```






```
// O operador + para cadeias de caracteres não faz uma soma, mas uma operação
// chamada concatenação, que combina a segunda cadeira após a primeira.
console.log("Olá, " + seu_nome + "!")
```

```
Olá, Igor!
```

```
// Usar o acento grave (`) permite adicionar variáveis dentro do texto, após
// um cifrão e entre chaves.
console.log(`Você tem ${sua_idade} anos.`)
```

```
Você tem 30 anos.
```

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

MÓDULO I

Parabéns! Você Já Pode Escrever Qualquer Programa em JavaScript

Acredite ou não, você já sabe tudo de que precisa sobre a sintaxe de JavaScript para escrever qualquer programa. Você sabe, inclusive, que pode procurar informações sobre como a linguagem funciona utilizando a **MDN**. Para isso, basta usar a ferramenta de busca da página ou procurar em seu motor favorito de busca por "**mdn** nome do recurso".

Ainda há muito mais para aprender sobre a linguagem, mas o exposto anteriormente basta para escrever qualquer programa. Não é brincadeira, é fato. É possível, inclusive, construir computadores completos que definem *uma única* instrução (*conhecido como one-instruction set computer ou OISC*).

A questão é que saber a sintaxe de uma linguagem é como conhecer palavras de um idioma e regras para construção de frases. Conhecer palavras e como combiná-las em frases não significa ter, necessariamente, o conhecimento necessário para escrever um artigo científico ou romance.

Analogamente, programar não é saber a sintaxe nem as regras de uma linguagem, mas saber *resolver problemas* usando a linguagem. Mais importante que conhecer a sintaxe é entender lógica de programação e desenvolver habilidades de *pensamento computacional*.

MÓDULO I

Assim, reafirmo que linguagens de programação são ferramentas. **Domine o pensamento computacional e você irá programar em qualquer linguagem de programação**. Para isso, bastará consultar a sintaxe da linguagem escolhida e procurar por comandos e subrotinas desejados.

Cada passo a seu tempo. Com o conhecimento da sintaxe uma primeira linguagem de programação, o objetivo acaba de tornar-se, portanto, aprender e desenvolver habilidades de pensamento computacional. Ou seja, aprender a pensar como um programador ou uma programadora para resolver problemas usando um computador.

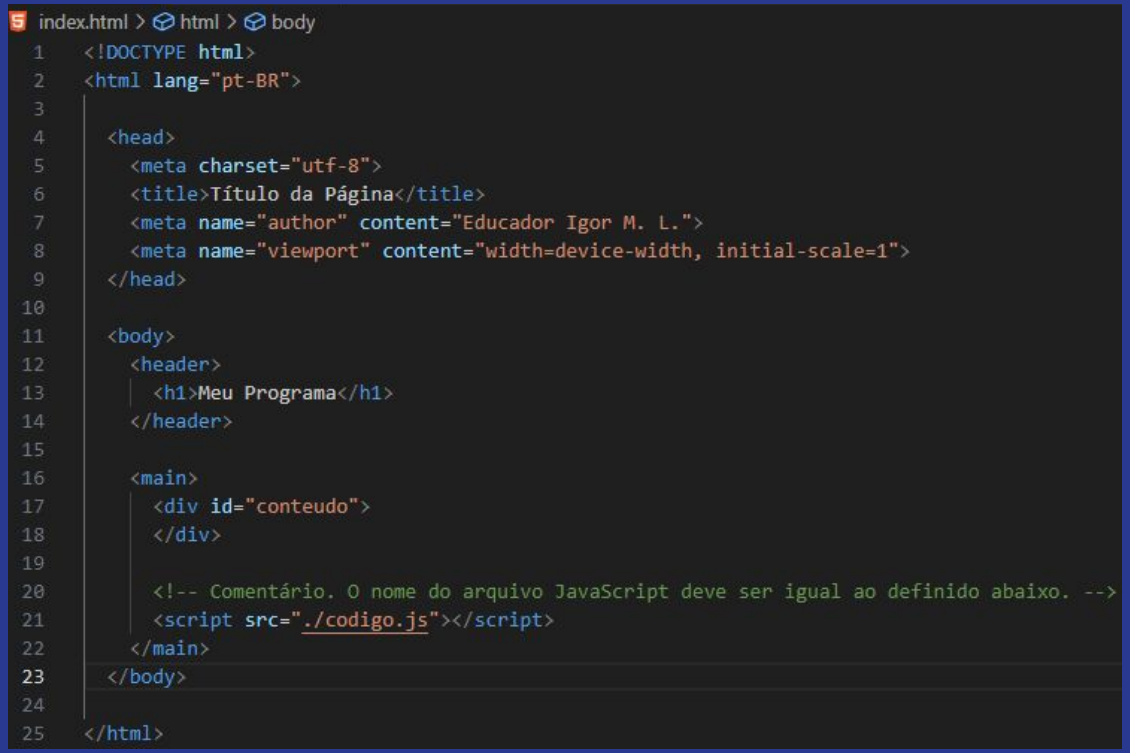
Contudo, antes de prosseguir, é útil aprender a escrever arquivos (*scripts*) com código-fonte. Afinal, você provavelmente não gostaria de ter que digitar um mesmo código toda vez que quiser usá-lo. Não faça o que uma máquina pode fazer por você.

MÓDULO I

Arquivos de Código-Fonte

JavaScript como primeira linguagem. **HTML** como segunda linguagem.
JavaScript é uma linguagem imperativa. Você comanda o interpretador a fazer o que seu programa deve desempenhar.
HTML é uma linguagem de marcação declarativa. Você declara ao interpretador o que gostaria que aparecesse em seu programa. Para isso, você usa *tags* para definir elementos que devam aparecer em sua página.
Assim, programar em **HTML** é diferente de programar em **JavaScript**, como se pode observar no bloco de código **HTML** a seguir.
Crie um arquivo de texto com o nome que quiser, mas com a extensão **.html**. Por exemplo, **website.html**.
O nome tradicional para a página inicial de um site (*ou diretório de um site*) é **index.html**.

MÓDULO I



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

MÓDULO I

A estrutura do código HTML é diferente da usada no código **JavaScript**. Simplificadamente, você:

1. Informou o tipo do arquivo (*DOCTYPE html*);
2. Declarou um documento iniciado em **<html>** e terminado em **</html>**.
   Tudo que está entre as duas *tags* faz parte do documento.
3. Declarou um cabeçalho (**<head>**), com título, nome do autor e instruções de como o navegador deverá exibir sua página;
4. Declarou um corpo (**<body>**) com o conteúdo. O corpo possui um cabeçalho (**<header>**) com um título de primeiro nível **<h1>**. Ele também possui uma região (** **) chamada conteúdo, que está vazia.
   O final do corpo inclui um arquivo **JavaScript** chamado **codigo.js**, que se criará a seguir.

MÓDULO I

Agora crie um arquivo chamado **codigo.js** (*caso o nome seja diferente, você deverá alterar o nome no código **HTML** anterior*) no mesmo diretório em que você criou o arquivo com sua página da Internet.

```
console.log("Esta mensagem aparecerá no console, não na página.")
```

Abra a página em seu navegador de Internet. Em seguida, abra o console de seu navegador (*aperte **F12***).
O texto da mensagem estará lá. Ou seja, de agora em diante, você poderá criar seus programas e salvá-los em um arquivo para, posteriormente, executá-los no navegador.

MÓDULO I

Criando Conteúdo para sua Página com JavaScript

Embora a solução anterior seja suficiente, é possível fazer melhor. Combinando **JavaScript** com **HTML** é possível escrever diretamente na página criada. Para isso, deve-se criar um novo elemento e adicioná-lo ao **Modelo de Objeto de Documentos** (*Document Object Model ou DOM) no arquivo* ***codigo.js****;*

O código a seguir emprega os seguintes recursos (a saber, funções, métodos e propriedades):

- document.getElementById() ;
- document.createElement() ;
- Element.innerHTML ;
- Node.appendChild() .

```
function adicione_elemento(valor, nome_elemento = "p") {
    const pai = document.getElementById("conteudo")
    const novo_elemento = document.createElement(nome_elemento)
    novo_elemento.innerHTML = valor
    pai.appendChild(novo_elemento)
}
```

MÓDULO I

O bloco de código anterior define uma função chamada **adicione_elemento()** que insere um valor em um elemento a ser criado. Neste momento, não se preocupe com a definição da função, pois é mais relevante aprender a usá-la.

Todavia, algo a observar é que o nome da função utiliza um verbo no modo imperativo (*adicione*). Novamente, você comanda seu programa a fazer algo em linguagens imperativas. Portanto, é comum definir nomes de funções com verbos cujas conjugações estão no modo imperativo. Em inglês, a função poderia chamar **add_element()**. Contudo, assim como nomes de variáveis e constantes, o nome de funções é de sua escolha (*desde que seja válido*). Se você quiser, você pode chamar a função de **xyz()**; desde que não mude a definição do corpo (*o código entre as chaves*), ela funcionará da mesma forma. Evidentemente, é melhor escolher um nome que informe o que o código faz ao invés de uma seqüência aleatória de caracteres.

MÓDULO I

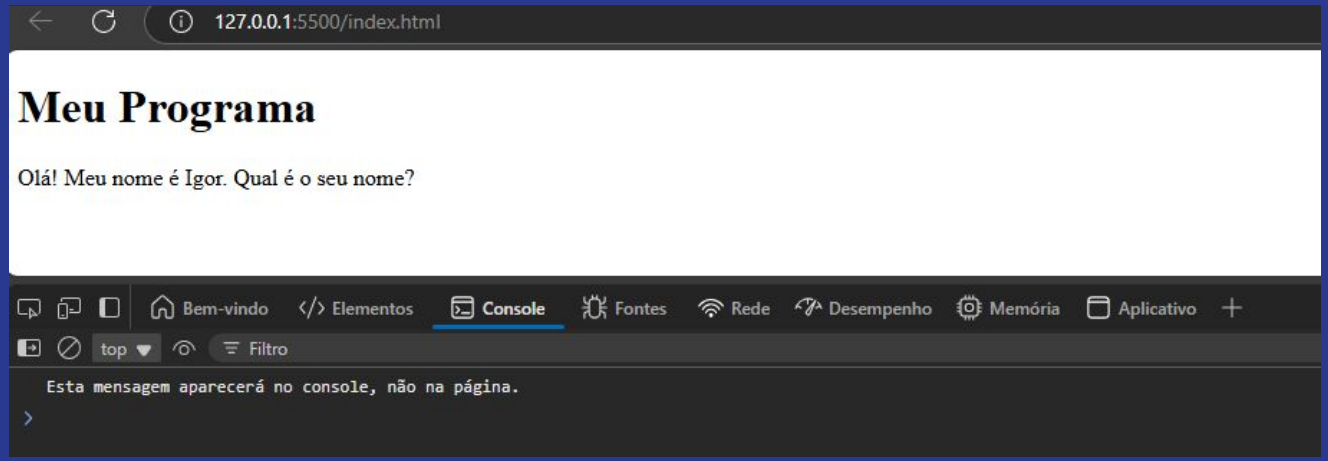
Para escrever na sua página usando código **JavaScript**, tudo que você precisará fazer é usar **adicione_elemento()** e passar um valor. Por exemplo, após definir a função, pode-se escrever:

JS codigo.js > ...

```
1   // Código antigo.
2   console.log("Esta mensagem aparecerá no console, não na página.")
3
4   function adicione_elemento(valor, nome_elemento = "p") {
5       const pai = document.getElementById("conteudo")
6       const novo_elemento = document.createElement(nome_elemento)
7       novo_elemento.innerHTML = valor
8       pai.appendChild(novo_elemento)
9   }
10
11
12
13
14  // Código novo.
15  adicione_elemento("Olá! Meu nome é Igor. Qual é o seu nome?")
```

MÓDULO I

Abra a página HTML criada anteriormente no navegador.
O que aparecerá no console? O que aparecerá na página?



MÓDULO I

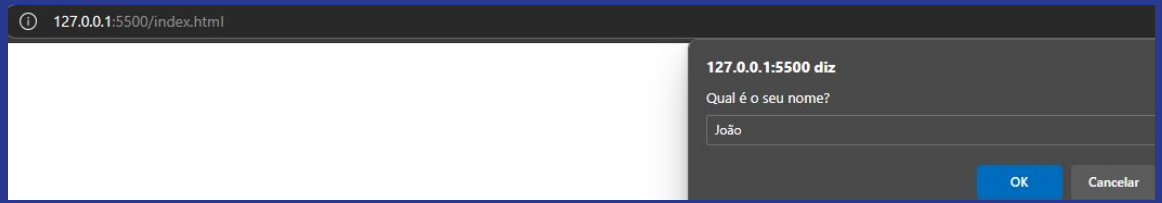
Criando Outros Elementos

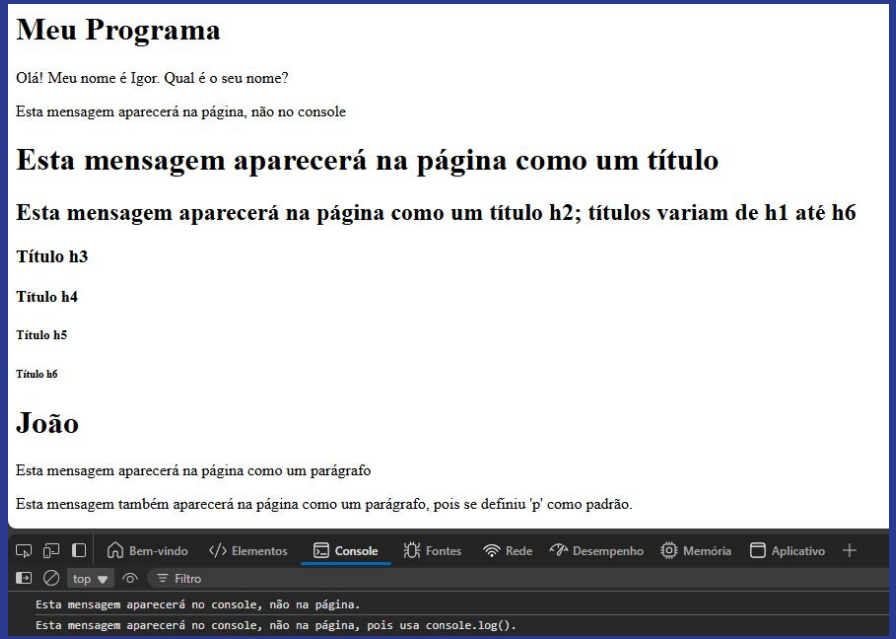
Toda vez que você salvar o arquivo de código **JavaScript** e atualizar a página no navegador (*por exemplo, usando **F5***), o conteúdo escrito usando **adicione_elemento()** aparecerá como texto no seu navegador. Assim, adicione uma nova linha definida no bloco de código a seguir por vez.
Salve o arquivo **JavaScript**, recarregue a página **HTML** no navegador, verifique os resultados.

codigo.js > ...

```
1  console.log("Esta mensagem aparecerá no console, não na página.")
2
3  function adicione_elemento(valor, nome_elemento = "p") {
4      const pai = document.getElementById("conteudo")
5      const novo_elemento = document.createElement(nome_elemento)
6      novo_elemento.innerHTML = valor
7      pai.appendChild(novo_elemento)
8  }
9
10 adicione_elemento("Olá! Meu nome é Igor. Qual é o seu nome?")
11
12 // Código novo.
13 // Você pode usar a função adicione_elemento() para escrever na sua página usando JavaScript.
14 adicione_elemento("Esta mensagem aparecerá na página, não no console")
15 // Títulos podem variar de h1 até h6.
16 adicione_elemento("Esta mensagem aparecerá na página como um título", "h1")
17 adicione_elemento("Esta mensagem aparecerá na página como um título h2; títulos variam de h1 até h6", "h2")
18 adicione_elemento("Título h3", "h3")
19 adicione_elemento("Título h4", "h4")
20 adicione_elemento("Título h5", "h5")
21 adicione_elemento("Título h6", "h6")
22 // Exemplo de entrada e armazenamento de valor lido em variável.
23 let nome = prompt("Qual é o seu nome?")
24 adicione_elemento(nome, "h1")
25 // Exemplo de criação de parágrafo.
26 adicione_elemento("Esta mensagem aparecerá na página como um parágrafo", "p")
27 adicione_elemento("Esta mensagem também aparecerá na página como um parágrafo, pois se definiu 'p' como padrão.")
28 // O retorno ao tradicional console.
29 console.log("Esta mensagem aparecerá no console, não na página, pois usa console.log().")
```

MÓDULO I





Em muitas linguagens de programação, criar conteúdo multimídia em um programa é mais complexo que em **JavaScript**, pois (*normalmente*) depende de bibliotecas externas. Em **JavaScript**, basta embutir um *script* (*arquivo com código*) em uma página escrita em **HTML** e abri-la em um navegador.

MÓDULO I

JavaScript e o Ponto e Vírgula

Existem linguagens de programação que demarcam o final de uma linha de código por um símbolo especial. Diversas linguagens, incluindo-se **C, C++** e **Java**, adotam um ponto e vírgula (**;**) para esse propósito.
Em **JavaScript**, o uso de um ponto e vírgula ao final de uma linha de código é opcional. No passado, ele era obrigatório, mas navegadores aceitavam código sem o uso de ponto e vírgula. Por vezes, isso gerava código incorreto.
Embora o problema seja mais raro atualmente, ele ainda existe em certo casos. Em geral, problemas podem acontecer quando uma linha começa com **parênteses** (**()**), **colchetes** (**[]**) ou **chaves** (**{}**), porque o interpretador pode considerar que seja a continuação da linha anterior. Por isso, existem pessoas e guias de estilo que recomendam o uso de ponto e vírgula ao final de cada linha de código.
Se quiser evitar possíveis problemas, termine todas as suas linhas de código usando ponto e vírgula. Quando se aprende a programar, inclusive, essa pode ser a melhor alternativa para evitar surpresas.

MÓDULO I

Se preferir, você omitir o ponto e vírgula do final de cada linha. Via de regra, o código funcionará corretamente.

O mais importante é testar seu código sempre, para garantir que funcione como o esperado. Com testes freqüentes, você identificará problemas assim que eles surgirem, potencialmente facilitando a correção da causa.

JS codigo.js > ...

```
1  console.log("Esta mensagem aparecerá no console, não na página.");
2
3  function adicione_elemento(valor, nome_elemento = "p") {
4      const pai = document.getElementById("conteudo");
5      const novo_elemento = document.createElement(nome_elemento);
6      novo_elemento.innerHTML = valor;
7      pai.appendChild(novo_elemento);
8  }
9
10 adicione_elemento("Olá! Meu nome é Igor. Qual é o seu nome?");
11
12 // Você pode usar a função adicione_elemento() para escrever na sua página usando JavaScript.
13 adicione_elemento("Esta mensagem aparecerá na página, não no console");
14 // Títulos podem variar de h1 até h6.
15 adicione_elemento("Esta mensagem aparecerá na página como um título", "h1");
16 adicione_elemento("Esta mensagem aparecerá na página como um título h2; títulos variam de h1 até h6", "h2");
17 adicione_elemento("Título h3", "h3");
18 adicione_elemento("Título h4", "h4");
19 adicione_elemento("Título h5", "h5");
20 adicione_elemento("Título h6", "h6");
21 // Exemplo de entrada e armazenamento de valor lido em varíavel.
22 let nome = prompt("Qual é o seu nome?");
23 adicione_elemento(nome, "h1");
24 // Exemplo de criação de parágrafo.
25 adicione_elemento("Esta mensagem aparecerá na página como um parágrafo", "p");
26 adicione_elemento("Esta mensagem também aparecerá na página como um parágrafo, pois se 'p' definiu 'p' como padrão.");
27 // O retorno ao tradicional console.
28 console.log("Esta mensagem aparecerá no console, não na página, pois usa console.log().");
```

MÓDULO I

Sua Página Com Estilo

Uma boa prática de programação é separar conteúdo de apresentação. Para a programação Web, isso pode ser feito com facilidade usando **CSS** para definir estilos para *tags* escritas **HTML** ou geradas via **JavaScript**.

Assim, para melhorar a aparência estética de sua página em construção, você pode criar um terceiro arquivo com a extensão .css no mesmo diretório dos **arquivos .js** e **.css**. Nesta seção, adota-se o nome **estilos.css** para o arquivo **CSS**.

No bloco de código **CSS**, deve-se notar que body, header, main, h1, h2, h3, h4, h5, h6 e p são *tags* usadas no arquivo **HTML** definido anteriormente.

MÓDULO I

estilos.css > p

```css
body {
    background: #fdf6e3;
    color: #929e04;
    font-family: sans-serif;
    line-height: 1.5;
    margin: 0;
    overflow-x: hidden;
    padding: 0;
}

header,
main {
    margin: 0 auto;
    max-width: 600px;
    padding: 3rem 1rem;
}

h1,
h2,
h3,
h4,
h5,
h6 {
    color: #7491d2;
    text-align: center;
    line-height: var(--line-height);
}

p {
    margin: 0.75rem 0;
    padding: 0;
}
```

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

MÓDULO I

Os valores escolhidos para cada parâmetro são arbitrários; Você pode alterá-los conforme suas preferências (*especialmente para cores e alinhamentos*). Os cores utilizam valores **HTML**, também conhecidos como cores em valores hexadecimais (***hex***). Você pode procurar por "**html color codes**" ou abrir um editor de imagem qualquer (*por exemplo, Paint no Windows; Gimp em qualquer sistema operacional; o KDE possui um programa para seleção de cores exibidas na tela*) para obter valores hexadecimais de cores. Alternativamente, a Wikipédia possui páginas com listas de cores em vários sistemas de cores:

Para aplicar o estilo na página, basta alterar o código **HTML** da seguinte forma: dentro de **<head>**, deve-se acrescentar a linha *<link rel="stylesheet" href="./estilos.css">*. Caso seu arquivo **CSS** tenha outro nome, basta modificar estilos.css para nome-do-seu-arquivo.css.

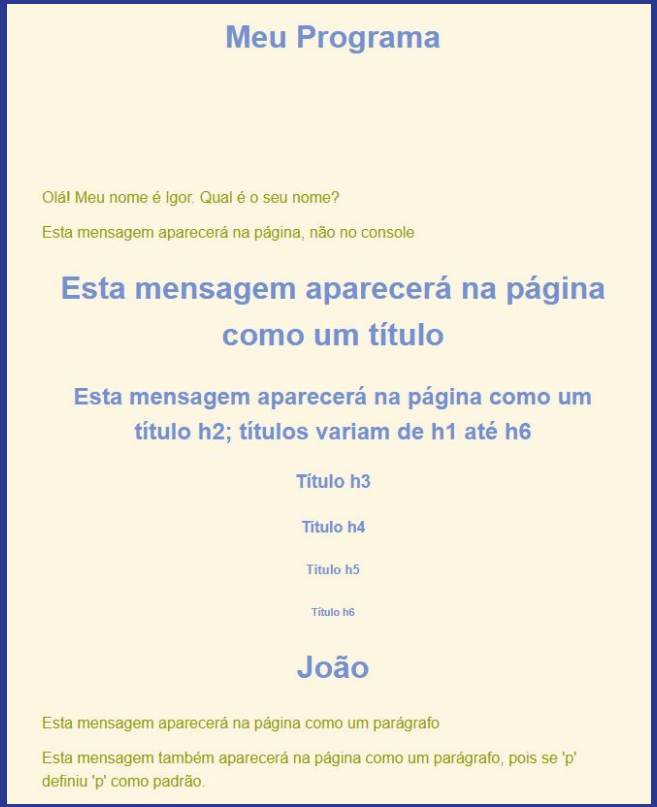
MÓDULO I

index.html > html

```html
<!DOCTYPE html>
<html lang="pt-BR">

  <head>
    <meta charset="utf-8">
    <title>Título da Página</title>
    <meta name="author" content="Educador Igor M. L.">
    <meta name="viewport" content="width=device-width, initial-scale=1">
<!-- Estilo aplicado dentro da teg head -->
    <link rel="stylesheet" href="./estilos.css">

  </head>

  <body>
    <header>
      <h1>Meu Programa</h1>
    </header>

    <main>
      <div id="conteudo">
      </div>

      <!-- Comentário. O nome do arquivo JavaScript deve ser igual ao definido abaixo. -->
      <script src="./codigo.js"></script>
    </main>
  </body>

</html>
```

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

MÓDULO I



Ao recarregar a página no navegador, ela estará com uma nova aparência.

MÓDULO I

JavaScript Como Linguagem Única Para Tudo

Existe uma frase famosa em países de língua inglesa que diz que "*se tudo que você tem é um martelo, tudo se parece com um prego*". Embora possa parecer um absurdo para o mundo real, algumas pessoas gostam de usar uma única ferramenta de programação para tudo. De uma perspectiva teórica, não há nada de errado com a escolha. Entretanto, é uma limitação e algo que também limita o crescimento pessoal.

Para quem deseja evoluir como profissional, na minha opinião, linguagens de programação são ferramentas. O aprendizado de outras linguagens e, sobretudo, de outros paradigmas de programação altera a forma de se pensar em problemas e soluções usando computadores.

Limitar-se a uma única linguagem de programação é como tentar cavar com um martelo. É possível? Sim, mas exige grandes esforços e tempo. Que tal uma pá?

MÓDULO I

Não se limite. JavaScript nem sempre é a melhor opção, assim como não existe uma linguagem melhor que todas as outras para todos os problemas. Não se deve fazer algo apenas por ser possível. É válido para aprendizado, mas existem contrapartidas que se deve considerar.

Com um repertório de ferramentas, pode-se escolher a melhor para cada problema. Em desenvolvimento de software, existem casos em que a solução mais simples pode ser a melhor. A elegância em programação comumente resulta de simplicidade aliada a boas idéias e práticas. Para cada problema, escolha a linguagem de programação capaz de expressar sua solução da forma mais simples possível.

MÓDULO I

EXERCÍCIOS...

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

MÓDULO I

FIM...

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