

Introdução ao JavaScript

Disciplina: Desenvolvimento Web

Prof. Dr. Rafael Will M. de Araujo



Conteúdo

1 Introdução

Introdução ao JavaScript (JS)

- Linguagem de programação criada pela Netscape em 1995:
 - ▷ Validação de formulários no lado cliente;
 - ▷ Interação com a página;
 - ▷ Alteração de comportamento da página;
- Foi feita como uma linguagem de *script*:
 - ▷ Linguagens de *script* são normalmente interpretadas e geralmente utilizadas para complementar programas complexos.
- Apesar do nome, não tem relação com a linguagem Java;
 - ▷ Mas a sintaxe é semelhante a do Java (como também C#, C, PHP e outras).

Introdução ao JavaScript (JS)

- É interpretada, ao invés de compilada;
- **Dinamicamente** tipada (uma variável pode assumir vários tipos de dados ao longo de um programa), e **fracamente** tipada (permite operações entre tipos de dados diferentes sem provocar erros).
 - ▷ A linguagem Python, por exemplo, também é dinamicamente tipada, mas com tipagem forte.
- Suporta expressões regulares (importante para validação de dados e outras tarefas com *strings*).
- É *case sensitive* (diferencia letras maiúsculas e minúsculas).

Introdução ao JavaScript (JS)

- É interpretada, ao invés de compilada;
- **Dinamicamente** tipada (uma variável pode assumir vários tipos de dados ao longo de um programa), e **fracamente** tipada (permite operações entre tipos de dados diferentes sem provocar erros).
 - ▷ A linguagem Python, por exemplo, também é dinamicamente tipada, mas com tipagem forte.
- Suporta expressões regulares (importante para validação de dados e outras tarefas com *strings*).
- É *case sensitive* (diferencia letras maiúsculas e minúsculas).
- Complementa as tecnologias do lado do **Cliente**:
 - ▷ **JavaScript**: Comportamento
 - ▷ **CSS**: Apresentação
 - ▷ **HTML**: Estrutura do documento
- CSS + HTML + JavaScript = DHTML (Dynamic HTML)

Introdução ao JavaScript (JS)

- O arquivo JS é um arquivo de texto com extensão .js
- Neste arquivo são declarados:
 - ▷ **Variáveis:** definem valores e armazenam dados;
 - ▷ **Funções:** definem comportamentos e ações para a página web;
 - ▷ **Eventos:** funções específicas disparadas a partir da interação do usuário com a página;
- O arquivo JS é basicamente uma sequência de comandos JavaScript. Cada comando é executado pelo navegador na sequência em que aparece no arquivo.

Integração com o HTML

- Inserindo JavaScript no HTML: código dentro do HTML (não recomendado):

Inserindo JavaScript diretamente no HTML

```
1 <script type="text/javascript">  
2     /* código JavaScript */  
3 </script>
```

- Importando o arquivo .js (recomendado):

Importando JavaScript de um arquivo .js externo

```
1 <script type="text/javascript" src="nome_arquivo.js">  
2 </script>
```

OBS: O atributo *type* não é mais obrigatório na *tag script*.

Integração com o HTML: a *tag script*

- Qualquer código JS (dentro da página ou um arquivo externo) deve ficar entre os marcadores `<script>` `</script>`.
- É um marcador HTML que **DEVE** ter a abertura e o fechamento (mesmo se for uma referência a um arquivo externo).
- Normalmente `<script></script>` é inserido no cabeçalho da página (dentro da *tag* `<head>` `</head>`), mas também pode ser colocado no corpo (entre a *tag* `<body>` `</body>`).
- Atributos comuns:
 - ▷ *type*: informa que o script é um JS (não obrigatório no HTML5) (padrão: `text/javascript`);
 - ▷ *src*: informa a localização do arquivo JS;
 - ▷ *async*: ativa a execução assíncrona;
 - ▷ *defer*: se o script executa apenas quando a página acabar de carregar.

Exemplo da *tag script* com alguns atributos

```
1 <script src="arquivo.js" defer async>
2 </script>
```


Integração com o HTML

- Ao carregar uma página, o navegador executa o seu código de cima para baixo;
 - ▷ Isso inclui também a leitura do código JS, como acontece com o CSS.
- Ou seja, tudo que for colocado em `/* código JavaScript */` (exemplo visto anteriormente) ou que estiver dentro de um arquivo .js será lido;
- Quando utilizamos um arquivo JS externo, o navegador fará uma requisição ao servidor (usando o método GET) para retornar o recurso.
 - ▷ Portanto, os arquivos ou códigos JS são incluídos como um anexo à sua página HTML.
- Assim como HTML e CSS, JS é um código executado no lado **cliente**.

Conteúdo

1 Introdução

2 Primeiros passos

3 Sintaxe básica

"Olá mundo"

Olá mundo!

```
1 <script>
2   alert("Olá Mundo");
3 </script>
4 <h1>Primeiro código em JS</h1>
```

- Comando **alert**: função que exibe uma janela com uma mensagem no navegador. Nenhuma navegação é permitida enquanto o *alert* não for fechado.
- Observe que o JS é executado **antes do navegador exibir o HTML**.

"Olá mundo" com arquivo externo

Arquivo *meu_script.js*

```
1 alert("Ola mundo!");
```

Olá mundo!

```
1 <script src="meu_script.js"></script>  
2 <h1>Primeiro código em JS</h1>
```

- O arquivo JS é basicamente uma sequência de comandos JavaScript.
- Cada comando é executado pelo navegador na sequência em que é escrito.
- Um comando pode ser terminado por ponto e vírgula (;). Não é obrigatório, mas é uma boa prática.
 - ▷ Outras linguagens com sintaxe parecida (C, C#, Java, etc) acusarão erro de sintaxe.

Definição de variáveis

- As variáveis em JS são “*containers*” para armazenar informação.
- Para declarar uma variável, devemos usar uma **palavra chave** e o **nome dessa variável**:

Declarando uma variável chamada x

1 `var x;`

- Em JS não há tipos para declarar a variável, apenas uma palavra chave **var**;
- Além da palavra chave **var**, existem os identificadores **let** e **const** para variáveis, cada uma com a sua função.

Definição de variáveis

- O identificador **var** existe desde o começo do JavaScript. Ela possui um problema de vazamento de escopo (veja sobre o *hoisting*):
<https://medium.com/opensanca/hoisting-em-javascript-9f22b1f78448>, portanto recomenda-se usar sempre os identificadores **let** e **const**;
- **let**: variáveis com escopo em bloco;
- **const**: variáveis de referência constante (uma vez definidas, não podem ter o seu valor alterado ao longo do programa).

Conteúdo

1 Introdução

2 Primeiros passos

3 Sintaxe básica

Sintaxe: variáveis e valores

- Quando criada, uma variável recebe valor **undefined**, utilizado como valor primitivo vazio.
- Existe também o valor **null**, mas esse é utilizado para zerar intencionalmente uma referência qualquer.
- Para atribuir um valor a variável, basta utilizar o operador de atribuição:

Declarando uma variável com valor inicializado

```
1 let x = 10;
```

- Strings em JS podem ser atribuídas com aspas simples ou aspas duplas (mas é preferível que sejam usadas aspas duplas!)

Declarando uma *string*

```
1 let x = "Aprender JS é legal";
```


Exibindo valores

- É possível exibir valores de variáveis no console do navegador (utilizado para auxiliar no desenvolvimento), através do comando `console.log()`:

Exibindo o valor de uma variável no console

```
1 let x = 10;  
2 console.log(x);
```

- Também é possível exibir esses valores no documento (HTML), através do comando `document.write()`:

Exibindo o valor de uma variável no documento HTML

```
1 let x = 10;  
2 document.write(x);
```

Sintaxe: variáveis e valores

- É possível concatenar strings, números e variáveis em JS, utilizando o operador `+`, resultando em uma string.

Concatenação de strings

```
1 let x = "JavaScript";  
2 let y = "surgiu em";  
3 let z = 1995;  
4 let resultado = "JavaScript" + "surgiu em" + 1995;
```

- Os tipos básicos do JavaScript são:
 - ▷ **Strings**: `let nome = "Professor"`
 - ▷ **Números (number)**, seja inteiro ou decimal: `let idade = 32` ou `let preco = 34.56`
 - ▷ **Booleano (boolean)**: `let verdade = true` ou `let mentira = false`
- O tipo de uma variável pode ser consultado com o operador `typeof(VARIAVEL)`:

Consultando o tipo de uma variável

```
1 let x = "JavaScript";  
2 console.log(typeof(x)); // imprime "string"  
  
3 let y = 12;  
4 console.log(typeof(y)); // imprime "number"  
  
5 let z = true;  
6 console.log(typeof(z)); // imprime "boolean"
```

Operadores aritméticos, lógicos e relacionais

Funcionam como a maioria das linguagens de programação:

- Operadores aritméticos: +, -, *, /, %
- Operadores lógicos: && (e), || (ou), ! (não)
- Operadores de comparação (relacionais): ==, !=, <=, <, >=, >
- Atribuição: =
- Os operadores de comparação possuem um detalhe extra: o JavaScript **tende a comparar tudo como se fosse string**.
- Para comparar se dois valores são iguais e do mesmo tipo de dado, devemos utilizar o operador relacional de **igualdade estrita**: ===

Arrays/vetores

- JS também suporta o uso de *arrays* (vetores). Para criar uma variável de *array*, utilizamos:

Criando arrays vazios

```
1 let alunos = new Array();  
2 /* ou então: */  
3 let alunos = [];
```

- Depois, basta inserir os valores nas posições do array:

Adicionando valores ao array

```
1 alunos[0] = "Aluno1";  
2 alunos[1] = "Aluno2";
```

- Forma alternativa:

Inicializando um array com valores

```
1 let alunos = ["Aluno1", "Aluno2"];
```

- Repare que não é preciso declarar o tamanho do *array*.

Arrays/vetores

- É possível adicionar elementos ao final do array através do método *push()*:
 - ▷ O método *push()* devolve o novo tamanho do array.

Adicionando um valor ao final do array

```
1 let alunos = ["Ana", "Pedro"];
2 tamanho = alunos.push("João");
3 console.log(tamanho);    // imprime 3
```

- Também é possível remover um elemento do final do array com o método *pop()*:
 - ▷ O método *pop()* devolve o elemento removido do array.

Removendo o último valor do array

```
1 let alunos = ["Ana", "Pedro", "Vinicius"];
2 valor = alunos.pop();
3 console.log(valor);    // imprime "Vinicius"
```

- A propriedade *length* devolve o tamanho do array:

Obtendo o tamanho do array

```
1 let alunos = ["Ana", "Bia", "Carlos", "Diego", "Eduardo"];
2 console.log(alunos.length);    // imprime 5
```

Objetos (JSON)

- O JavaScript possui uma estrutura similar aos dicionários do Python, o JSON (*JavaScript Object Notation*).
- Para criar um JSON podemos fazer:

Criando objetos

```
1 let objeto = new Object();  
2 // ou de maneira simplificada:  
3 let objeto = {};
```

- Um JSON é uma coleção de pares chave-valor, onde a chave deve ser uma *string* e o valor pode ser **qualquer outro tipo válido** do JavaScript.
- Para acessar um valor no JSON, podemos usar a notação ponto (.) ou colchetes ([]):

Acessando atributos (chaves)

```
1 objeto.atributo // devolve o valor associado à chave "atributo" no objeto  
2 objeto["atributo"] // devolve o valor associado à chave "atributo" no objeto
```

Funções

- Assim como a maioria das linguagens de programação, o JavaScript permite o uso de funções para deixar o código mais organizado.
- Uma função nada mais é do que um bloco de código executado quando é chamada.
 - ▶ **Por exemplo:** executar uma função quando clicar em um botão

Sintaxe básica de uma função

```
1 function nome_da_funcao() {  
2     alert("Olá mundo!");  
3 }
```

- Funções em JS podem receber argumentos, separados por vírgula:

Função com parâmetros

```
1 function funcao(arg1, arg2) {  
2     alert(arg1);  
3     alert(arg2);  
4 }
```

Funções: devolvendo (retornando) valores

- Funções em JS não declaram tipo de retorno. Entretanto, podem devolver (retornar) algum valor para quem chamou;
- Para isso, utiliza-se a palavra chave **return**:

Função com retorno

```
1 function funcao(){  
2     var x=5;  
3     return x;  
4 }
```

- Para executar a função, basta usar o nome e os parênteses:

Chamando uma função

```
1 funcao();
```


Funções anônimas

- Também é possível construir funções sem nome no JavaScript (**funções anônimas**).
- Como as funções são tratadas como objetos no JavaScript, elas podem ser guardadas em variáveis:

Função anônima

```
1  const minhaFuncao = function () {  
2      return (5 + 3);  
3  }  
4  minhaFuncao();
```

- Funções anônimas podem ser criadas associando-as diretamente a uma variável, como também a um parâmetro de outra função (veremos mais à frente).

Fluxo - Estrutura de Decisão (if/else)

- O JavaScript também define estruturas de controle (**if...else if...else**).
- O uso é quase idêntico à maioria das linguagens de programação:

Bloco if/else: sintaxe básica

```
1  if (condicao1) {  
2      // código executado se a condicao1 é verdadeira  
3  } else if (condicao2) {  
4      //código executado se a condicao2 é verdadeira  
5  } else {  
6      //código executado se nenhuma das condições acima é verdadeira  
7  }
```

Fluxo - Estrutura de Decisão (switch)

- Outra estrutura de controle em JavaScript é o **switch**:
- O **switch** avalia uma expressão, combinando o valor da expressão para uma cláusula **case** (caso), e executa as instruções associadas ao **case**. Funciona da seguinte maneira:
 - ▷ A expressão switch é avaliada uma vez;
 - ▷ O valor da expressão é comparado com os valores de cada caso;
 - ▷ Se houver correspondência, o bloco de código associado é executado;
 - ▷ Se não houver correspondência, o bloco de código padrão é executado.

Bloco switch: sintaxe básica

```
1  switch(n) {  
2      case 1:  
3          // bloco 1  
4          break;  
5      case 2:  
6          // bloco 2  
7          break;  
8      default:  
9          // caso nenhum bloco seja executado  
10 }
```

Fluxo - Estrutura de decisão (switch)

switch/case: exemplo prático

```
1 let dia = 2
2 let dia_semana;
3 switch(dia) {
4     case 1:
5         dia_semana = "Domingo";
6         break;
7     case 2:
8         dia_semana = "Segunda-feira";
9         break;
10    case 3:
11        dia_semana = "Terça-feira";
12        break;
13    case 4:
14        dia_semana = "Quarta-feira";
15        break;
16    case 5:
17        dia_semana = "Quinta-feira";
18        break;
19    case 6:
20        dia_semana = "Sexta-feira";
21        break;
22    case 7:
23        dia_semana = "Sábado";
24        break;
25    default:
26        dia_semana = "Inválido";
27 }
28 alert(dia_semana);
```

Fluxo - Estruturas de Repetição

- O JavaScript também define estruturas de repetição, muito parecido com outras linguagens de programação (C, C++, C#, Java, etc).
- Bloco **while**: verifica a expressão lógica antes. Enquanto ela for verdadeira, executa o bloco de código.

Estrutura de repetição: while

```
1 let i = 1;  
2 while (i <= 5) {  
3     alert(i);  
4     i = i + 1;  
5 }
```

Comando **for**: (<valor inicial>, <expressão lógica>, <incremento>):

Estrutura de repetição: for

```
1 for (let i=0; i<5; i++) {  
2     alert(i);  
3 }
```

- Bloco **do...while**: executa pelo menos uma vez, e só ao final verifica a expressão lógica.

Estrutura de repetição: do/while

```
1 i = 1  
2 do {  
3     alert(i);  
4     i = i + 1;  
5 } while (i <= 5);
```