

Instituto de

computação



Desenvolvimento Web JavaScript

Prof. Leonardo Cruz.

leonardocruz@id.uff.br

Departamento de Computação, UFF

Introdução

- O código **em uma página** pode ser concebido em três visões distintas:
 - Estrutura e conteúdo: **HTML**
 - Apresentação: **CSS**
 - Comportamento: **JavaScript**
- Vantagens:
 - Reuso de partes do projeto
 - Modularidade
 - Flexibilidade e facilidade de manutenção
 - Legibilidade

Características - javaScript

- É uma linguagem poderosa, com sua grande aplicação (não é a única) do lado cliente (browser)
- É uma linguagem de scripts que permite interatividade nas páginas web
- É incluída na página HTML e interpretada pelo navegador
- É simples, porém pode-se criar construções complexas e criativas

Características - javaScript



JavaScript **NÃO** é JAVA

Características - JavaScript

O que podemos fazer

- Validar entrada de dados em formulários: campos não preenchidos ou preenchidos incorretamente poderão ser verificados
- Realizar operações matemáticas e computação
- Abrir janelas do navegador, trocar informações entre janelas, manipular com propriedades como histórico, barra de status...

Características - JavaScript

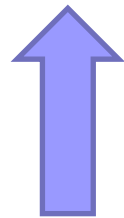
O que podemos fazer

- Interagir com o conteúdo do documento tratando toda a página como uma estrutura de objetos
- Interagir com o usuário através do tratamento de eventos

Formas de Organização - Uso

- O código javaScript pode ser organizado de três formas diferentes:
- Dentro próprio código HTML:

`Diga alô`



evento (ao clicar...)

Exemplo

```
<html>
<head>
<title>Alo!</title>
</head>
<body>
  corpo do html
  <br>
  <a href="#" onclick="alert('Oi mundo!')">Diga oi</a>
</body>
</html>
```

veja: ex1.html

Formas de Organização - Uso

- Separado em uma tag de script (dentro da tag `<head></head>` ou `<body></body>`):

```
<script type="text/javascript">
```

```
    alert("alo mundo") ;
```

```
</script>
```

Exemplo

```
<html>
```

```
<head>
```

```
<title>Alo!</title>
```

```
<script type="text/javascript">
```

```
    alert("alo mundo") ;
```

```
</script>
```

```
</head>
```

```
<body>
```

```
    corpo do html
```

```
</body>
```

```
</html>
```

veja: ex2.html

Formas de Organização - Uso

- Dentro de um arquivo “texto” com extensão .js sendo chamado por uma tag script:

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

Observação

- Vamos encontrar várias páginas que colocam o javascript no head, isso causa bloqueio no carregamento da página (fica uma página alguns segundos em branco ou parcialmente carregada)
- Para evitar esse bloqueio, coloque o JavaScript o mais pra baixo que puder; se possível, logo antes de fechar o body

Exemplo

```
<html>  
<head>  
<title>Alo!</title>  
</head>  
<body>  
  corpo  
  <script type="text/javascript" src =“alomundo.js”> </script>  
</body>  
</html>
```

Observação

- Quando em produção minifique o arquivo javascript
- Minificar é tornar um arquivo menos pesado removendo seus trechos desnecessários para que o código rode mais rápido.

Exemplo

```
var nome;  
nome = prompt("Qual é seu nome?");  
alert("Olá, " + nome);
```

“minificado”

```
var nome;nome=prompt("Qual é seu nome?"),alert("Olá, "+nome);
```

alomundo.**min**.js

<https://jscompress.com/>

Sintaxe do Javascript

- Tudo é case-sensitive, ou seja: **teste** é diferente de **Teste**
- Construções simples: após cada instrução, finaliza-se utilizando um ponto-e-vírgula:
Instrução1;
Instrução2;
- Ex:
alert("alo");
alert("mundo");

- Comentários de uma linha:

`alert("teste"); // comentário de uma linha`

- Comentário de várias linhas:

`/* este é um comentário
de mais de uma linhas */`

- Saída de dados: em lugar de usar a função `alert`, podemos utilizar:

`document.write("<h1>teste</h1>");`

□ Onde **document** representa a própria página e `write` escreve no seu corpo.

Exemplo

<html>

<head>

<title>Alo!</title>

</head>

<body>

corpo do html

<script type="text/javascript">

document.write("<h1>oi</h1>");

</script>

</body>

</html>

veja ex3.html

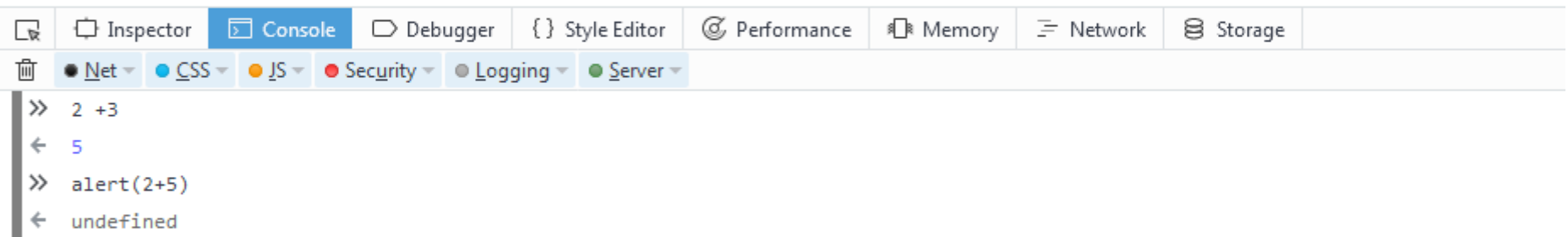
Console do navegador

- O console nos permite **testar códigos** diretamente no navegador sem termos que colocar uma tag `<script>` na página.
- no Google Chrome o console pode ser acessado por meio do atalho **Control + Shift + C**; no Firefox, pelo atalho **Control + Shift + K**

Console do navegador

corpo do html

oi



Variáveis

- Variáveis são usadas para armazenar valores temporários
- Usamos a palavra reservada **var** para defini-las
- Em JS, as variáveis são fracamente tipadas, ou seja, o tipo não é definido explicitamente e sim a partir de uma atribuição (=)

■ Ex:

var x = 4; ← Declaração e atribuição de valor

var y; ← Declaração sem atribuição

y = 2; ← Atribuição

alert (x + y);

Variáveis

■ Números: inteiros e decimais:

var i = 3;

var peso = 65.5;

var inteiroNegativo = -3;

var realNegativo = -498.90;

var expressao = 2 + (4*2 + 20/4) - 3;

■ Strings ou cadeia de caracteres:

var nome = "josé";

var endereco = "rua" + " das flores"; ← concatenação

nome = nome + " maria"; ← concatenação

endereco = "rua a, numero " + 3;

Escrevendo no Console do navegador

- Sem usar o **alert**
- `var mensagem = "Olá mundo";`
`console.log(mensagem);`

Exemplo

```
<html>
```

```
<head>
```

```
<title>Alo!</title>
```

```
</head>
```

```
<body>
```

corpo do html

```
<script type="text/javascript">
```

```
var a = 5;
```

```
var b = 12;
```

```
var c = a + b;
```

```
console.log(c);
```

```
</script>
```

```
</body>
```

```
</html>
```

veja ex4.html (abrir console)

Tipo Lógico

- Lógico: tipo que pode ter os valores **true** ou **false**

```
var aprovado = true;  
alert(aprovado);
```

Tipo String

- `var nome = "Leo";`
- `nome.length;` // tamanho da string
`nome.replace("e","a");` // retorna laio
- **String é imutável.**
- se a variável `nome` for impressa após a chamada da função `replace` o valor continuará sendo "Leo"

Tipo String

- `var nome = "Leo";`
- `nome.length;` // tamanho da string
`nome.replace("e","a");` // retorna lao
- **`nome = nome.replace("e","a");`**

Conversão: String - Número

```
var textoInteiro = "10";  
var inteiro = parseInt(textoInteiro);  
var textoFloat = "10.22";  
var float = parseFloat(textoFloat);
```

Exemplo: toFixed

```
var milNumber = 1000;  
var milString = milNumber.toFixed(2);  
console.log(milString); // imprime a string "1000.00"
```

altera o número de casas decimais com a função
toFixed

Array

- Arrays: alternativa para o armazenamento de conjuntos de valores:

```
var numeros = [1,3,5];
```

```
var strNumeros = [];
```

```
strNumeros[0] = "Primeiro";
```

```
strNumeros[1] = "Segundo";
```

```
var cidades = [ ];
```

```
cidades[0] = "Rio de janeiro";
```

```
cidades[1] = "Niterói";
```

```
cidades[2] = "Maricá";
```

```
alert("A capital do RJ é " + cidades[1]);
```

Array

- Tamanho de um array: usamos a propriedade **length** do próprio array
`alert(cidades.length);`
- Último item de um array:
`alert(cidades[cidades.length-1]);`

Array Associativo

- baseados também na ideia `array[indice] = valor`
- O índice/chave de um array associativo é geralmente uma string

```
var idades = [];
```

```
idades["leo"] = 29;
```

```
idades["pedro"] = 20;
```

```
idades["carlos"] = 20;
```

```
alert("Minha idade é: " + idades["leo"]);
```


- **typeof**: inspecionar o tipo de uma variável ou valor:

```
var a = "teste";
```

```
alert( typeof a); // string
```

```
alert( typeof 95.8); // number
```

```
alert( typeof 5); // number
```

```
alert( typeof false); // boolean
```

```
alert( typeof true); // boolean
```

```
alert( typeof null); // object
```

```
var b;
```

```
alert( typeof b); // undefined
```

Operador de tipos

- Utilizando typeof podemos ter os seguintes resultados:
 - ☐ undefined: se o tipo for indefinido.
 - ☐ boolean: se o tipo for lógico
 - ☐ number: se for um tipo numérico (inteiro ou ponto flutuante)
 - ☐ string: se for uma string
 - ☐ object: se for uma referência de tipos (objeto) ou tipo nulo

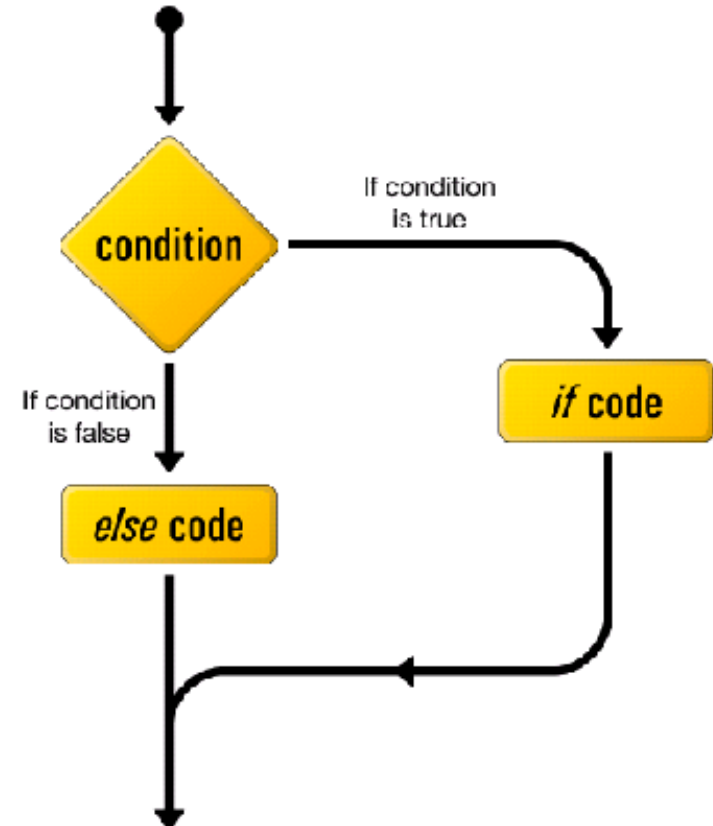
Estruturas de decisão – if e else

■ Sintaxe:

```
if (condição) {  
    código da condição verdadeira;  
}  
else {  
    código da condição falsa;  
}
```

{ simboliza um início/begin

} representa um fim/end



Operadores condicionais e lógicos

| | |
|----|--------|
| > | A > B |
| >= | A >= B |
| < | A < B |
| <= | A <= B |
| == | A == B |
| != | A != B |


← A é igual a B

← A é diferente de B

- || : or
- ! : not

Estruturas de decisão – if e else

```
if (a == 1) {  
    alert("sim");  
} else {  
    alert("não");  
}
```



Estruturas de decisão – Switch

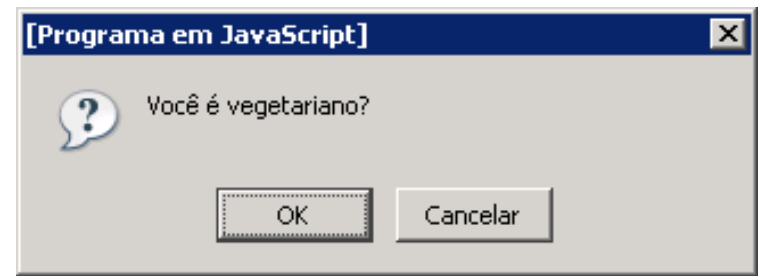
```
switch (expressão) {  
    case valor 1:  
        //código a ser executado se a expressão = valor 1;  
        break;  
    case valor 2:  
        //código a ser executado se a expressão = valor 2;  
        break;  
    ...  
    case valor n:  
        //código a ser executado se a expressão = valor n;  
        break;  
    default:  
        //executado caso a expressão não seja nenhum dos valores;  
}
```

Janelas de diálogo - Confirmação

- Nos permite exibir uma janela pop up com dois botões: **ok** e **cancel**
- Funciona como uma função:
 - Se o usuário clicar em **ok**, ela retorna **true**; em **cancel** retorna **false**

■ Ex:

```
var vegetariano = confirm("Você é vegetariano?");
if (vegetariano == true) {
    alert("Coma mais proteínas");
}
else {
    alert("Coma menos gordura");
}
```



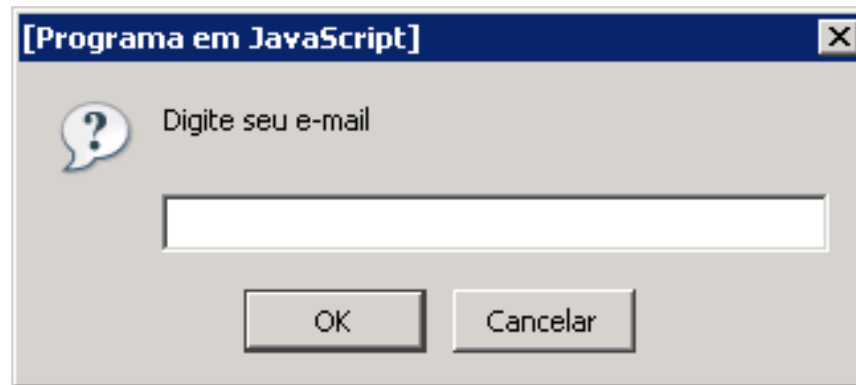
Janelas de diálogo - Prompt

- Nos permite exibir uma janela pop up com dois botões (**ok** e **cancel**) e uma caixa de texto
- Funciona como uma função: se o usuário clicar em **ok** e **prencher a caixa de texto**, ela retorna **o valor do texto**; em **cancel** retorna **null**
- O segundo parâmetro pode ser preenchido como uma sugestão

Janelas de diálogo - Prompt

■ Ex:

```
var email = prompt("Digite seu e-mail","");  
alert("O email " + email + " será usado para  
spam.");
```



Janelas de diálogo - Prompt

- O que lemos da janela prompt é uma string
- Podemos converter strings para inteiro utilizando as funções pré-definida **parseInt** e **parseFloat**
- **parseInt(valor, base)**: converte uma string para inteiro.
 - O valor será convertido para inteiro e base é o número da base (vamos usar base 10)
- **parseFloat(valor)**: converte uma string para um valor real/ponto flutuante

Janelas de diálogo - Prompt

■ Ex:

```
var notaStr = prompt("Qual a sua nota?","");  
var trabStr = prompt("Qual o valor do trabalho?","");  
var nota = parseFloat(notaStr,10);  
var trab = parseFloat(trabStr,10);  
nota = nota + trab;  
  
alert("Sua nota é: " + nota );
```

Estrutura de repetição - for

- Executa um trecho de código por uma quantidade específica de vezes

- Sintaxe:

```
for (inicio; condição; incremento/decremento) {  
    //código a ser executado.  
}
```

- Ex:

```
var numeros = [1, 2, 3, 4, 5];  
for (var i = 0; i < numeros.length; i++) {  
    numeros[i] = numeros[i] * 2;  
    document.write(numeros[i] + "<br/>");  
}
```

Expressões compactadas

- Em JS podemos utilizar formas “compactada” instruções:

numero = numero + 1 equivale a numero++

numero = numero - 1 equivale a numero--

numero = numero + 1 equivale a numero += 1

numero = numero - 1 equivale a numero -= 1

numero = numero * 2 equivale a numero *= 2

numero = numero / 2 equivale a numero /= 2

Estrutura de repetição - while

- Executa um trecho de código enquanto uma condição for verdadeira
- Sintaxe:

```
while (condicao) {  
    //código a ser executado  
}
```

Ex:

```
var numero = 1;  
while (numero <= 5) {  
    alert("O número atual é: " + numero);  
    numero = numero + 1;  
}
```

Estrutura de repetição – do...while

- Executa um trecho de código enquanto uma condição for verdadeira
- Mesmo que a condição seja falsa, o código é executado pelo menos uma vez
- Sintaxe:

```
do {  
    //código a ser executado.  
} while (numero <= 5) ;
```

Ex:

```
var numero = 1;  
do {  
    alert("O número atual é: " + numero);  
    numero = numero + 1;  
} while (numero <= 5) ;
```