



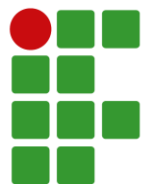
**Licenciatura  
em Computação**

Curso de Licenciatura em Computação  
Disciplina de Programação I  
Professor Tauã M. Cabreira

# AULA 16

---

Eventos e Validação de Formulários



**INSTITUTO FEDERAL**

Sul-rio-grandense  
Câmpus Pelotas

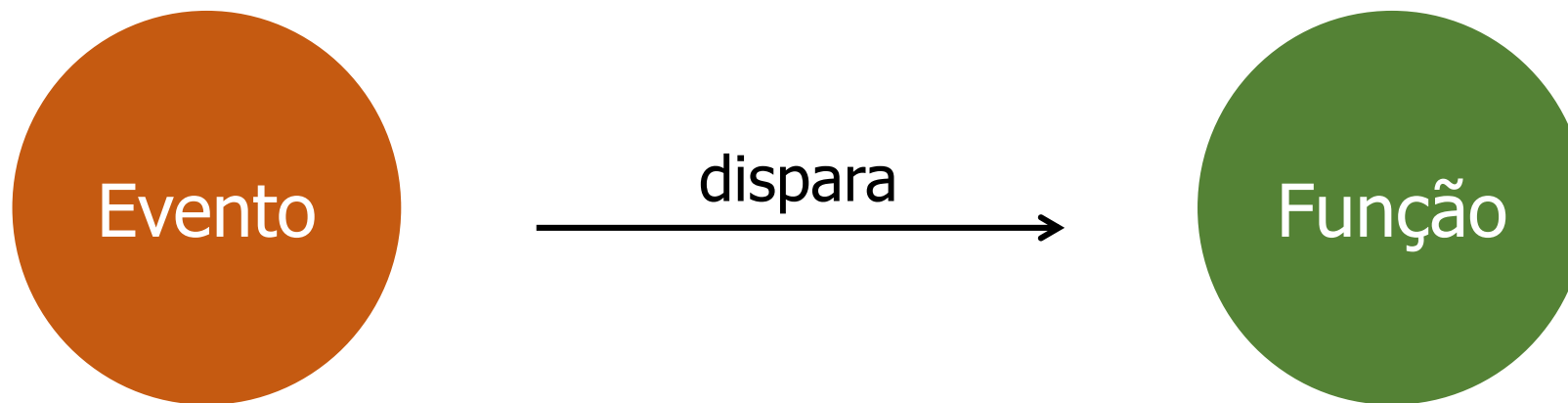
# Introdução

---

- Sem eventos, não há JavaScript! A ocorrência de um **evento** dispara uma **ação**, isto é, faz um **script** funcionar.
- Toda vez que ocorre **interação** com um documento web, um **evento** é disparado.
- Eventos podem ocorrer por interferência do **usuário** (clicar em um link) ou por interferência do próprio **navegador** (carregar uma página).

# JavaScript

---



- **Ações em elementos HTML**

- Carregamento da página
- Clique em um botão
- Campo de formulário modificado

- **Bloco de código executado quando é "chamado"**

- Abre uma janela de alerta
- Modifica o árvore do documento HTML
- Modifica o estilo de um elemento HTML

# Reagindo a Eventos

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      // Código JavaScript
    </script>
  </head>
  <body>
    <h1 onclick="this.innerHTML = 'Opaaaaaa!'">Clique neste texto!</h1>
  </body>
</html>
```

O trecho de código JavaScript foi inserido diretamente no HTML. O atributo HTML **onclick** atrela este evento ao elemento **h1**. Toda vez que ele é clicado, a propriedade **innerHTML** do próprio elemento (this) é modificada.



# Reagindo a Eventos

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<script>
```

```
    function mudaTexto(id) {
```

```
        id.innerHTML = "Opaaaaaaa!";
```

```
    }
```

```
</script>
```

```
</head>
```

```
<body>
```

```
    <h1 onclick="mudaTexto(this);">Clique neste texto!</h1>
```

```
</body>
```

```
</html>
```

A função recebe como parâmetro o próprio elemento que a chama (this).

Agora temos uma função **mudaTexto** atrelada ao evento **onclick**.

# Eventos + HTML DOM

---


- Os eventos podem ser atrelados a elementos HTML diretamente através dos atributos de marcação: onclick, onmouseover, etc.
- O HTML DOM também possibilita atrelar eventos a elementos HTML através do JavaScript.

# Eventos + HTML DOM

```
<!DOCTYPE html>
<html>
  <head></head>
  <body>
    <h1 id="bloco">Clique neste texto!</h1>

    <script>
      document.getElementById("bloco").onclick = mudaTexto;
      function mudaTexto() {
        document.getElementById("bloco").innerHTML = "Opaaaaaaa!";
      }
    </script>
  </body>
</html>
```

Usando o HTML DOM, pode-se atrelar o evento **onclick** ao elemento com a ID **bloco** e chamar a função **mudaTexto**.



# Manipulador de Eventos

---



# Fundamentos

---

- **Manipulador de evento** (*event handler*): Função a ser executada quando um evento é disparado.
- **Disparador de evento**: Elemento HTML ao qual foi atrelado um manipulador de evento.
- **Manipulador-padrão de evento** (*default event handler*): Função-padrão do navegador executada quando ocorrem determinados eventos.

# Manipulador-padrão

---

- Submeter um **formulário** dispara uma função-padrão que faz com que o navegador renderize a página definida no **action**.
- Toda vez que se atrela um **manipulador de evento** a um elemento HTML, ele é executado antes do **manipulador-padrão**.
- Pode-se, então, cancelar a ação padrão do navegador para um evento com o uso da declaração **return false**.

# Atrelando Eventos: Exemplo 1

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      function mudaTexto(id){
        id.innerHTML = "Opaaaaaaa!";
      }
    </script>
  </head>
  <body>
    <h1 onclick="mudaTexto(this);">Clique neste texto!</h1>
  </body>
</html>
```

Esta solução deve ser evitada, pois contraria o princípio da separação das camadas de desenvolvimento. Não se deve misturar **marcação** com **script**, pois dificulta a manutenção.

# Atrelando Eventos: Exemplo 2

```
<!DOCTYPE html>
<html>
  <head></head>
  <body>
    <h1 id="bloco">Clique neste texto!</h1>

    <script>
      document.getElementById("bloco").onclick = mudaTexto;
      function mudaTexto() {
        document.getElementById("bloco").innerHTML = "Opaaaaaaa!";
      }
    </script>
  </body>
</html>
```

Aqui temos HTML e JavaScript separado em camadas  
(considere sempre que este trecho JS deve estar em um  
arquivo separado!)

QUAL É O PROBLEMA!?

# Atrelando Eventos: Exemplo 3

```
<!DOCTYPE html>
<html>
  <head></head>
  <body>
    <h1 id="bloco">Clique neste texto!</h1>

    <script>
      var elemento = document.getElementById("bloco");
      elemento.addEventListener("click", mudaTexto);
      elemento.addEventListener("click", disparaJanela);

      function mudaTexto() {
        this.innerHTML = "Opaaaaaaa!";
      }

      function disparaJanela() {
        window.alert("Olá Mundo!");
      }
    </script>
  </body>
</html>
```

Através do método **addEventListener** é possível atrelar vários eventos ao mesmo elemento e acionar diversas funções.

**JS NA FORMA CORRETA!**

# Atrelando Eventos

---

- **addEventListener(evento, função, booleano)**
  - O primeiro parâmetro é o tipo de evento ("click", "mousedown").
  - O segundo parâmetro é a função que queremos chamar quando o evento ocorre.
  - O terceiro parâmetro é um valor booleano opcional, normalmente **falso**.

# Tipos de Eventos

---

- Os eventos JS podem ser agrupados em categorias:
  - Eventos de mouse
  - Eventos de teclado
  - Eventos de formulário
  - Eventos de página
- E existem ainda muitos outros tipos de evento:
  - [http://www.w3schools.com/jsref/dom\\_obj\\_event.asp](http://www.w3schools.com/jsref/dom_obj_event.asp)

# Eventos de mouse

Evento	Descrição
click	Ocorre quando o usuário pressiona o botão esquerdo do mouse ou a tecla Enter sobre um elemento.
dblclick	Ocorre quando o usuário pressiona seguidamente duas vezes o botão esquerdo.
mousedown	Ocorre quando o usuário pressiona qualquer um dos botões do mouse.
mouseup	Ocorre quando o usuário solta o botão do mouse.
mouseover	Ocorre quando o usuário desloca o ponteiro do mouse sobre um elemento.
mouseout	Ocorre quando o usuário desloca o ponteiro para fora de um elemento.
mousemove	Ocorre quando o usuário move o ponteiro.



# Eventos `mouseover` e `mouseout`



```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
    <script src="script.js"></script>
```

```
    <link href="estilo.css" type="text/css" rel="stylesheet">
```

```
  </head>
```

```
  <body>
```

```
    <h1 id="bloco">Passe o mouse</h1>
```

```
  </body>
```

```
</html>
```

Os arquivos de estilo CSS e de script JS são linkados no <head> do document HTML.



# Eventos **mouseover** e **mouseout**

---

```
#bloco{  
    width: 120px;  
    padding: 20px;  
    background-color: #ba170b;  
    font: 16px Arial;  
    color: #fff;  
    text-align: center;  
}
```



Estilização CSS aplicada ao elemento  
h1 presente no document HTML

# Eventos `mouseover` e `mouseout`

```
window.onload = function(){  
    var elemento = document.getElementById("bloco");  
  
    elemento.addEventListener("mouseover", moveSobre);  
    elemento.addEventListener("mouseout", moveParaFora);  
}
```

O elemento com a ID **bloco** tem dois eventos atrelados a ele: **mouseover** e **mouseout**.

Cada evento dispara uma função diferente.

```
function moveSobre(){  
    this.innerHTML = "Obrigado!";  
}
```

← Duas funções são criadas para modificar o texto com a propriedade **innerHTML**.

```
function moveParaFora(){  
    this.innerHTML = "Passe o mouse";  
}
```

← A palavra-chave **this** refere-se ao elemento que recebeu o evento.

# Eventos `mousedown` e `mouseup`

```
window.onload = function(){  
    var elemento = document.getElementById("bloco");  
  
    elemento.addEventListener("mousedown", pressionaMouse);  
    elemento.addEventListener("mouseup", soltaMouse);  
}
```

O elemento com a ID **bloco** tem dois eventos atrelados a ele: **`mousedown`** e **`mouseup`**.

Cada evento dispara uma função diferente.

```
function pressionaMouse(){  
    this.style.backgroundColor = "#335387";  
    this.innerHTML = "Me Solta!";  
}
```

← Além de alterar o texto, as funções também modificam o estilo CSS do elemento HTML.

```
function soltaMouse(){  
    this.style.backgroundColor = "#338744";  
    this.innerHTML = "Obrigado!";  
}
```

Vamos usar os mesmos arquivos `exemplo.html` e `estilo.css`

# Eventos de teclado

Evento	Descrição
keydown	Ocorre quando o usuário pressiona uma <b>tecla qualquer</b> do teclado. Manter a tecla pressionada faz com que o evento se repita várias vezes.
keypress	Ocorre quando o usuário pressiona uma tecla do teclado que resulte em um <b>caractere</b> ou na <b>alteração de um texto</b> na tela. Manter a tecla pressionada faz com que o evento se repita várias vezes.
keyup	Ocorre quando o usuário solta uma tecla do teclado.

# Eventos de formulário

---

Evento	Descrição
blur	Ocorre quando um elemento perde o foco.
change	Ocorre quando um campo (input, textarea ou select) tem seu conteúdo alterado e perde o foco.
focus	Ocorre quando um elemento recebe o foco.
reset	Ocorre quando o botão <b>reset</b> de um formulário é clicado.
submit	Ocorre quando o botão <b>submit</b> de um formulário é clicado.

# Validação de Formulário

---

# Validação de Formulários

---

- Os eventos de teclado e de formulário podem ser utilizados para validação dos controles.
- Geralmente, a validação consiste em verificar se todos os campos de **preenchimento obrigatório** de um formulário possuem um valor válido.
- Pode-se ainda adicionar máscaras aos campos com **formatação específica**, tais como CPF e telefone. Estes campos também podem restringir a **entrada de dados**, aceitando apenas valores numéricos.



# Validação de Formulários

---

```
<form id="cadastro" action="exemplo.html" method="post" autocomplete="off">
  <fieldset>
    <legend>Formulário de Cadastro</legend>
    <label>Nome: <input type="text" name="nome"></label>
    <label>CPF: <input type="text" name="CPF"></label>
    <label>Telefone: <input type="text" name="telefone"></label>
    <label>Data de Nascimento: <input type="date" name="data"></label>
    <input type="submit" id="botao" name="botao" value="Enviar">
  </fieldset>
</form>
```

# Preenchimento obrigatório

O evento `onload` é ativado após o carregamento completo da página. Este evento dispara uma função anônima.



```
window.onload = function() {  
    var formulario = document.getElementById("cadastro");  
    formulario.addEventListener("submit", validaFormulario);  
}
```



O formulário com a ID **cadastro** é acessado através do método **getElementById**. Na sequência, o evento **submit** é atrelado ao formulário para disparar a função **validaFormulario**.

# Preenchimento obrigatório

```
function validaFormulario(event) {  
    formulario = document.getElementById("cadastro");  
    n = formulario.elements.length;  
  
    }  
}
```



A propriedade **elements** retorna uma coleção de controles que estão dentro do formulário.

Já a propriedade **length** retorna o número de controles do formulário.

# Preenchimento obrigatório

---

- Complete a função `validaFormulario` para que todos os campos tenham preenchimento obrigatório.
- Use uma estrutura de repetição para percorrer todos os controles do formulário - já sabes o número de controles ;)
- Cada controle pode ser acessado individualmente como se fosse um elemento de um array através de seu índice.

# Preenchimento obrigatório

---

- Use uma estrutura de seleção para verificar o atributo **value** de cada controle.
- Se ele estiver vazio, mostre uma janela de alerta indicando qual o campo deve ser preenchido. Como identificar o campo?  
Acessando o atributo **name**!
- Quando for identificado que o controle do formulário está vazio, coloque o foco no respectivo controle e estilize sua borda com a cor vermelha.

# Preenchimento obrigatório

---

Aqui vai a resposta!

# Objeto event

---

- Quando ocorre um evento, é criado um objeto **event**, que contém diversas informações relacionadas ao evento.
- É possível saber que tipo de evento foi acionado e qual elemento HTML disparou tal evento.
- Estas informações podem ser obtidas através das propriedades **type** e **target**.

# Aceitando apenas números


---

- A cada uma das teclas do teclado é atribuído um **código numérico** que permite identificá-la.
- Para teclas alfanuméricas, o **código da tecla** é igual ao valor decimal da **codificação ASCII** para letras minúsculas e números.
- A propriedade **keyCode** é utilizada para identificar o código de qualquer tecla pressionada no teclado.



# Aceitando apenas números

```
window.onload = function(){  
    var formulario = document.getElementById("cadastro");  
    formulario.addEventListener("submit", validaFormulario);  
  
    formulario.CPF.addEventListener("keypress", mascaraCPF);  
    formulario.telefone.addEventListener("keypress", mascaraFone);  
}
```



O evento **keypress** foi atrelado aos controles de CPF e de telefone. Quando este evento ocorre, as função **mascaraCPF** e **mascaraFone** são executadas.

# Aceitando apenas números

```
function mascaraCPF(event) {  
    alert(event.keyCode)  
}
```



A caixa de alerta irá exibir o código da tecla pressionada.

- Modifique a função **mascaraCPF** para verificar o código da tecla pressionada. Se o código estiver fora do intervalo correspondente aos números, use o método **preventDefault**.
- **Códigos das teclas?**
  - <http://www.cambiaresearch.com/articles/15/javascript-char-codes-key-codes>

# Aceitando apenas números

O tamanho do conteúdo digitado no controle também pode ser verificado através da propriedade **length**. O controle está limitado a 13 caracteres (incluindo . e -).

```
function mascaraCPF(event) {  
    if(event.keyCode < 48 || event.keyCode > 57 || this.value.length > 13) {  
        event.preventDefault();  
    }  
}
```

Se o código da tecla pressionada não estiver dentro do intervalo **48-57**, significa que não é um número.

Neste caso, o método **preventDefault** impede a ação natural do evento **keypress** que seria imprimir a tecla.

# Aplicando máscaras

---

- O conceito de máscara em JavaScript consiste em completar o controle de um formulário conforme a informação digitada pelo usuário.
- **Máscara de CPF:** completar com . (ponto) a cada três números e com – (traço) nos dois últimos. **Ex.: 000.000.000-00**
- **Máscara de Telefone:** completar com ( antes do primeiro número e com ) depois do segundo número.  
**Ex.: (53) 0000-0000**

# Aplicando máscaras

---

- Complete as funções `mascaraCPF` e `mascaraFone` para aplicar as máscaras correspondentes ao formato exigido.
- Toda vez que a função for acionada pelo evento **keypress**, deve-se verificar o tamanho do conteúdo do controle (use a propriedade **length**).
- Na sequência, deve-se concatenar o caractere correspondente (ponto ou traço ou parênteses) ao conteúdo do controle (use a propriedade **value**).

# Aplicando máscaras

---

Aqui vai a resposta!

# Aplicando máscaras

---

Aqui vai a resposta!



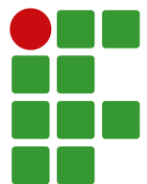
**Licenciatura  
em Computação**

Curso de Licenciatura em Computação  
Disciplina de Programação I  
Professor Tauã M. Cabreira

# AULA 16

---

Eventos e Validação de Formulários



**INSTITUTO FEDERAL**  
Sul-rio-grandense  
Câmpus Pelotas