

JavaScript

Módulo Básico





FUNCTIONS

Funções e Eventos

- **Funções**

- É possível **criar funções** com **passagem** de **parâmetros**, **criar funções anônimas**, fazer uma **função devolver** um **determinado valor** ou um **conjunto de valores** e, ainda, **utilizar uma função** como um **módulo** que contém um **trecho de código** que se **repete** e que pode ser **acionado** em **vários pontos do programa**
- Dessa forma, **cada vez** que **necessitamos executar** esse **trecho** de **código**, **chamamos o módulo**, sem a necessidade de **reescrever seus comandos**



Funções e Eventos

- **Eventos**

- Os **eventos** também podem ser *mais bem explorados*, pois há **vários outros** além do **submit** e **click**, como **change**, **blur**, **focus**, **keypress**...



Funções e Eventos

- **Modularização**

- Os **módulos** servem para melhor **organizar** nosso **código**, **permitir a reutilização** de **trechos de programa** e **facilitar a sua leitura e manutenção**
- O uso da **modularização** é igualmente **essencial** para a **organização do programa**
- **Exemplo:**
 - Você precisa **desenvolver** um **sistema** para uma empresa que deve **cadastrar** os **clientes**, **funcionários** e **estagiários**
 - Em todos esses **cadastros**, deve-se **validar** o **CPF** da pessoa
 - Se você **desenvolver** uma **função** para **validar** o **CPF**, ela poderá ser **utilizada** nos **três cadastros** desse **sistema**

Funções e Eventos

- **Observação:**
 - As *funções* são também chamadas de **métodos**, **procedures** e **módulos**
 - Com *pequenas variações de definição dependendo da linguagem e do paradigma de programação utilizado*



Funções e Eventos

- ***Functions e Arrow Functions***

- A partir do momento em que nossos **programas** vão **crescendo** em **tamanho** e **complexidade**, **dividi-los** em **blocos de código menores**, ou seja, em **funções**, apresenta **inúmeras vantagens**
- Pode-se destacar a possibilidade de **reaproveitamento** de **código**, a **melhor organização** e **maior facilidade** em **entender um problema grande dividindo-o** em **blocos menores** (**dividir** para **conquistar**), além dos **benefícios relacionados ao trabalho em equipe**

() => {}

Funções e Eventos

- **Functions e Arrow Functions**

- Uma **função** em **JS** pode ser **construída** com a **palavra reservada **function**** ou como uma **declaração de constante**
- **usando** uma **notação** conhecida como **arrow function** (**função seta**), em que a **função** é **atribuída** para uma **variável**

- **Sintaxe:**

```
<script>
function ola() {
  alert("Olá. Seja muito bem-vindo!")
}
ola()
</script>
```

```
<script>
const ola2 = () => {
  alert("Olá. Seja muito bem-vindo, novamente!")
}
ola2()
</script>
```


Funções e Eventos

- ***Functions e Arrow Functions***

- As ***vantagens*** da ***segunda forma*** estão relacionadas à ***proteção*** dada às ***const*** em ***JS*** e a uma ***sintaxe*** mais ***curta*** (***não*** necessita da ***cláusula return***) em ***funções*** que ***podem*** ser ***construídas*** com uma ***única atribuição***

- ***Exemplo:***

- Se executarmos um ***programa*** que ***contenha*** a ***declaração*** de duas ***functions*** com o ***mesmo nome***, o ***programa*** irá ***rodar***, ***sem acusar erro***
 - ***Caso isso aconteça a partir da declaração de const, a linguagem alerta para o erro no código***

Funções e Eventos

- **Functions e Arrow Functions**

- As vantagens da **segunda forma** estão relacionadas à **proteção dada às `const` em JS e a uma sintaxe mais curta**

Contudo, caso você se sinta mais confortável declarando os seus módulos de código com **function**, não há problema

- Há várias discussões na web sobre o tema. Alguns preferem continuar com **function** – pois ela permite uma leitura mais intuitiva do código.

Outros preferem **const** por permitir uma **sintaxe mais curta** e com a **proteção**

- **Caso isso aconteça a partir da declaração de `const`, a linguagem alerta para o erro no código**

Funções e Eventos

- **Funções com passagem de Parâmetros**
 - Os **parâmetros** permitem **ampliar** as funcionalidades da **função**
 - **Imaginem se tivéssemos uma função para exibir “Muito obrigado!”, e outra para exibir “Por favor, digite corretamente os dados”**
 - **Precisaríamos de inúmeras funções no programa**
 - **Em vez disso, passamos a mensagem a ser exibida pela função no momento em que a acionamos no programa**
 - **Chamar/invocar a função consiste em uma linha de código com o nome da função com os parâmetros inseridos dentro dos parênteses**

Funções e Eventos

- ***Funções com passagem de Parâmetros***

- ***Sintaxe do `alert()`***

// chama o método alert() passando o texto "Muito Obrigado!"

`alert("Muito Obrigado!")`



Funções e Eventos

- *Funções com passagem de Parâmetros*
 - *Crie uma pasta **css**, **img** e **js***
 - *Crie um arquivo intitulado de **estilos.css** com o conteúdo abaixo, e salve dentro da pasta **css**:*

```
h1 { border-bottom-style: inset; }
pre { font-size: 1.2em; }
img.normal { float: left; height: 300px; width: 300px; }
img.alt { float: left; height: 420px; width: 300px; }
img.exerc { float: left; height: 260px; width: 260px; }
span { margin-left: 70px; }
select { width: 150px; }
.detalhes { width: 380px; }
.oculta { display: none; }
.exibe { display: inline; }
.exibe-linha { display: block; }
/* para exibir no último exercício, o parágrafo em uma linha */
```

Funções e Eventos

- *Funções com passagem de Parâmetros*
 - *Exemplo (01):*
 - *Função com passagem de parâmetro*

```
8      <title>Exemplo 01 | bkBank Academy</title>
9      </head>
10     <body>
11     <script>
12         // função recebe 2 parâmetros: nota e media
13         const situacao = (nota, media) => {
14             if (nota >= media) {
15                 alert("Aprovado")
16             } else {
17                 alert("Reprovado")
18             }
19         }
20         const prova1 = Number(prompt("Qual Nota: ")) // lê uma nota
21
22         situacao(prova1, 7) // chama a função passando 2 parâmetros
23     </script>
24 </body>
25 </html>
```

Figura 1 – JS de função com passagem de parâmetro

Funções e Eventos

- **Funções com passagem de Parâmetros**

- **Exemplo (01):**

- **Função com passagem de parâmetro**

Observe que a função `situacao()` recebe os parâmetros `nota` e `media` – que devem estar entre parênteses após o sinal de “=”

O programa inicia pela leitura da nota do aluno. Em seguida, é realizada uma chamada da função

O conteúdo da variável `prova1` e o valor “7” são então passados para a função. Nela, esses valores são atribuídos para `nota` e `media`.

```
22     situacao(prova1, 7) // chama a função passando 2 parâmetros
23     </script>
24     </body>
25     </html>
```

Figura 1 – JS de função com passagem de parâmetro

Funções e Eventos

- **Funções com passagem de Parâmetros**
 - É comum criarmos funções com parâmetros contendo **valores default**, ou seja, se esse parâmetro não for informado, o **valor padrão** é atribuído para essa variável
 - A **média**, por exemplo, poderia conter o **valor default 7**. assim, a função pode receber apenas nota para todas as avaliações em que a **média** é **7**
 - Para avaliações com média **diferente** de **7**, deve-se, então, informar os **2 parâmetros**
 - A forma de indicar um valor padrão para um parâmetro é:

```
const situacao = (nota, media = 7) => { ... }
```


Funções e Eventos

- **Funções com passagem de Parâmetros**
 - Os termos **parâmetro** e **argumento** são utilizados para denominar as variáveis passadas no momento da chamada da função
 - Há uma pequena diferença entre eles: os nomes das variáveis (**nota** e **media**) são chamadas de **parâmetros**, já os valores reais desses parâmetros (o valor da **nota1** e o **valor 7**) são chamados de **argumentos** da função
 - Contudo, no geral, os termos **parâmetros** e **argumentos** são utilizados indistintamente

Funções e Eventos

- ***Funções com retorno de Valor***

- Para fazer uma **função** **retornar** um **valor**, utiliza-se o comando **return** seguido do **conteúdo de retorno**
- Para a **função do exemplo (01)** retorne um **valor indicativo da situação do aluno**, poderíamos **modifica-la da seguinte forma**:

```
const situacao = (nota, media) => {  
  const resultado = nota >= media ? "Aprovado" : "Reprovado"  
  return resultado  
}
```

Funções e Eventos

- ***Funções com retorno de Valor***

- ***É possível atribuir o retorno de uma função para uma variável e, em seguida, exibi-la na página***

```
const aluno1 = situacao(prova1, 7)  
resp.innerText = `Situação: ${aluno1}`
```

- ***É permitido utilizar o próprio retorno da função como parte de um cálculo ou como parâmetro de outro método***

```
alert(`A situação do aluno é: ${situacao(prova1, 7)}`)
```

Funções e Eventos

- *Funções com retorno de Valor*
 - *Functions com uma única atribuição, declaradas com **const** podem omitir o return*
 - *O valor atribuído a **const** é retornado por ela*
`const situacao = (nota, media) => (nota >= media ? "Aprovado" : "Reprovado")`
 - *Os parênteses depois da seta são opcionais*
 - *Caso a função contenha um único parâmetro, os parênteses envolvendo esse parâmetro também são opcionais*

Funções e Eventos

- ***Funções com retorno de Valor***

- ***Exemplo (02):***

- ***O programa da Revenda Avenida deverá ler o modelo, ano de fabricação e preço do veículo***
 - ***Na sequência, o programa deve classificar o veículo como “Novo” (do ano atual), “Seminovo” (até 2 anos de uso) ou “Usado”***
 - ***Também deve apresentar o valor da entrada e o saldo em 10x (sem juros)***
 - ***A entrada deve ser de 50% para veículos novos ou de 30% para veículos classificados como seminovos ou usados***
 - ***Para a classificação e o cálculo da entrada, serão utilizadas funções com retorno de valor***

Funções e Eventos

- *Funções com retorno de Valor*
 - *Exemplo (02):*



Revenda Avenida - Promoção

Modelo do Veículo:

Ano de Fabricação:

Preço R\$:

Sandero - Seminovo

Entrada R\$: 15000.00

+10x de R\$: 3500.00

Figura 2 – Renderização do HTML | Exemplo (02)

Funções e Eventos

- *Funções com retorno de Valor*
 - *Exemplo (02):*

```
1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <link rel="stylesheet" href="../css/estilos.css">
8   <title>Exemplo 02 | bkBank Academy</title>
9 </head>
10 <body>
11   
12   <h1> Revenda Avenida - Promoção </h1>
13   <form>
14     <p>Modelo do Veículo:
15       <input type="text" id="inModelo" required autofocus>
16     </p>
17     <p>Ano de Fabricação:
18       <input type="number" id="inAno" required>
19     </p>
20     <p>Preço R$:
21       <input type="number" min="0" step="0.01" id="inPreco" required>
22     </p>
23     <input type="submit" value="Classificar - Calcular Entrada e Parcelas">
24   </form>
25   <h3 id="outResp1"></h3>
26   <h3 id="outResp2"></h3>
27   <h3 id="outResp3"></h3>
28   <script src="../js/exemplo_2.js"></script>
29 </body>
30 </html>
```

Figura 2 – HTML do Exemplo (02)

Funções e Eventos

- *Funções com retorno de Valor*
 - *Exemplo (02): - Parte 1*

```
1  const frm = document.querySelector("form") // obtém elementos da página
2  const resp1 = document.querySelector("#outResp1")
3  const resp2 = document.querySelector("#outResp2")
4  const resp3 = document.querySelector("#outResp3")
5
6  frm.addEventListener("submit", (e) => {
7      e.preventDefault() // evita envio do form
8
9      const modelo = frm.inModelo.value // obtém o conteúdo dos campos
10     const ano = Number(frm.inAno.value)
11     const preco = Number(frm.inPreco.value)
12
13     const classificacao = classificarVeiculo(ano) // chama funções e atribui
14     const entrada = calcularEntrada(preco, classificacao) // ... retorno às variáveis
15
16     const parcela = (preco - entrada) / 10 // usa retorno da função para cálculo
17
18     resp1.innerText = modelo + " - " + classificacao // exibe as respostas
19     resp2.innerText = `Entrada R$: ${entrada.toFixed(2)}`
20     resp3.innerText = `+10x de R$: ${parcela.toFixed(2)}`
21 })
```

Figura 3 – JS do Exemplo (02) – Parte 1
Aula 07 | Módulo Básico

Funções e Eventos

- **Funções com retorno de Valor**

- **Exemplo (02): - Parte 1**

```
1  const frm = document.querySelector("form") // obtém elementos da página
2  const resp1 = document.querySelector("#outForm-1")
```

Observe que, após obter os dados do form, são feitas chamadas às funções `classificarVeiculo()` e `calcularEntrada()` com parâmetros sendo passados

O retorno dessas funções é atribuído para as variáveis `classificacao` e `entrada` que, na sequência, são exibidas na página

Ou seja, delegamos partes das tarefas desse programa para funções

```
17
18  resp1.innerHTML = modelo + " - " + classificacao // exibe as respostas
19  resp2.innerHTML = `Entrada R$: ${entrada.toFixed(2)}`
20  resp3.innerHTML = `+10x de R$: ${parcela.toFixed(2)}`
21  })
```

Figura 3 – JS do Exemplo (02) – Parte 1
Aula 07 | Módulo Básico

Funções e Eventos

- *Funções com retorno de Valor*
 - *Exemplo (02): - Parte 2*

```
23 // função recebe o ano do veículo como parâmetro
24 const classificarVeiculo = (ano) => {
25     const anoAtual = new Date().getFullYear() // obtém o ano atual
26     let classif
27     if (ano == anoAtual) { // condições para definir classificação do veículo
28         classif = "Novo"
29     } else if (ano == anoAtual - 1 || ano == anoAtual - 2) {
30         classif = "Seminovo"
31     } else {
32         classif = "Usado"
33     }
34     return classif // retorna a classificação
35 }
36
37 // função recebe valor e status do veículo como parâmetro
38 const calcularEntrada = (valor, status) =>
39     status == "Novo" ? valor * 0.5 : valor * 0.3
```

Figura 3 – JS do Exemplo (02) – Parte 2

Funções e Eventos

- **Funções com retorno de Valor**

- **Exemplo (02): - Parte 2**

```
23 // função recebe o ano do veículo como parâmetro
24 const classificarVeiculo = (ano) => {
```

A função `classificarVeiculo()` recebe um `ano` como `parâmetro`. Esse valor deve ser passado na chamada da função. A partir dele, realizam-se as comparações para atribuir à variável `classif` a indicação de que o veículo é `novo`, `seminovo` ou `usado`

No final, é necessário retornar essa variável para o programa que invocou a função

Esse valor será então atribuído no programa principal para a variável `classificacao`

Funções e Eventos

- **Funções com retorno de Valor**

- **Exemplo (02): - Parte 2**

```
23 // função recebe o ano do veículo como parâmetro
```

```
24 const classificarVeiculo = (ano) => {
```

Observe que agora são passados dois parâmetros: **valor** e **status**. Utilizamos esses nomes para destacar que os nomes dos parâmetros **não** precisam ser iguais aos nomes das variáveis do programa principal

E, como o retorno pode ser indicado a partir de uma única atribuição, não é necessário utilizar a palavra **return**

```
36
```

```
37 // função recebe valor e status do veículo como parâmetro
```

```
38 const calcularEntrada = (valor, status) =>
```

```
39   status == "Novo" ? valor * 0.5 : valor * 0.3
```

Figura 3 – JS do Exemplo (02) – Parte 2

Funções e Eventos

- ***Funções com parâmetros Rest***
 - ***O operador Rest (...) é utilizado para unir um conjunto de elementos em um vetor***
 - ***Ele também pode ser utilizado nas funções para receber um conjunto de parâmetros, que são convertidos para um vetor***



Funções e Eventos

- *Funções com parâmetros Rest*
 - *Exemplo (03):*
 - *Função com uso dos parâmetros Rest*

```
11 <script>
12   const calcularMedia = (...notas) => {
13     const num = notas.length;    // notas é um array
14     if (num == 0) {
15       console.log("Informe, no mínimo, uma nota");
16       return;
17     }
18     let soma = 0;                // vai acumular a soma das notas
19     for (nota of notas) {
20       soma += nota;              // soma o valor dos argumentos
21     }
22     const media = soma / num;    // calcula a media
23     console.log(`Média: ${media.toFixed(1)}`);
24   }
25   // exemplos de chamada de calcularMedia() com nº de argumentos diferentes
26   calcularMedia(6, 7, 8);       // Média: 7.0
27   calcularMedia(2, 10);         // Média: 6.0
28   calcularMedia(7.5, 10, 8, 9.5); // Média: 8.8
29   calcularMedia();              // Informe, no mínimo, uma nota
30   alert("Pressione F12 para ver no console as saídas do programa")
31 </script>
```

Figura 4 – JS do Exemplo (03)
Aula 07 | Módulo Básico

Funções e Eventos

- **Funções com parâmetros Rest**

- **Exemplo (03):**

A função inicia com um teste que exibe uma mensagem caso o número de argumentos seja igual a zero. Como o **operador Rest** cria um vetor com os parâmetros passados para a função, pode-se utilizar a **propriedade length** para recuperar o tamanho do vetor

Em seguida, é possível utilizar uma estrutura de repetição com o comando **for..of** para obter o valor de cada elemento passado na chamada da função

Observe que realizamos no programa várias chamadas à função, todas com diferente número de argumento

```
29     calcularMedia();           // Informe, no mínimo, uma nota
30     alert("Pressione F12 para ver no console as saídas do programa")
31 </script>
```

Figura 4 – JS do Exemplo (03)
Aula 07 | Módulo Básico

Funções e Eventos

- ***Funções com parâmetros Rest***

- *Esse processo de enviar vários parâmetros para serem trabalhados em uma função também pode ser realizado a partir da palavra reservada **arguments***
- *Com **arguments** é possível ter acesso a cada um dos argumentos passados à função (igualmente manipulados como elementos de vetor)*



Funções e Eventos

- **Funções Anônimas**

- As *funções anônimas* permitem *definir a programação* de um *bloco de código* **sem atribuir** um **nome** para a **função**
- Nos *programas desenvolvidos anteriormente*, essa *sintaxe* das **arrow functions** já estava *presente*, geralmente *associadas ao evento* **submit** do *form*
- **Exemplo:**

```
form.addEventListener("submit", (e) => {  
    const nome = frm.inNome.value  
    resp.innerText = `Olá ${nome}`  
    e.preventDefault()  
})
```

Funções e Eventos

- **Funções Anônimas**

- As *funções anônimas* permitem *definir a programação* de um *bloco de código* **sem atribuir** um **nome** para a **função**
- Nos *programas desenvolvidos anteriormente*, essa *sintaxe* das **arrow functions** já estava *presente*, geralmente *associadas ao evento* **submit** do **form**
- **Exemplo:**

Ao clicar no botão **submit** do **form** é executada uma função anônima que recebe "**e**" como parâmetro, seguida pela notação de uma **arrow function**

}

Funções e Eventos

- **Funções Anônimas**

- As **arrow functions** foram *introduzidas na ES6*
- *Permitem escrever funções com uma sintaxe mais enxuta*

```
let myFunction = (a, b) => a * b
```

- *Sintaxe **antes** da **arrow function***

```
hello = function() {  
  return "Hello World!";  
}
```

Funções e Eventos

- **Funções Anônimas**

- **Sintaxe *com arrow function***

```
hello = () => {  
  return "Hello World!";  
}
```

- ***Se a função tiver apenas uma instrução e a instrução retornar um valor, é permitido remover as chaves e a palavra-chave *return****

Funções e Eventos

- ***Funções Anônimas***

- ***Arrow function que retorna um valor por padrão***

- ```
hello = () => "Hello World!";
```

- ***Arrow function com parâmetro***

- ```
hello = (val) => "Hello " + val;
```

- ***Arrow function sem parâmetro***

- ```
hello = val => "Hello " + val;
```

# Funções e Eventos

- **Funções Anônimas**

- No próximo exemplo, utilizaremos o **método** `setInterval()` para **demonstrar** um **novo uso** para as **funções anônimas**
- O método `setInterval()` faz uma chamada de função a cada **intervalo de tempo**, indicado em milissegundos

```
const mostraHora = () => {
 const data = new Date()
 const hora = data.getHours()
 const min = data.getMinutes()
 const seg = data.getSeconds()
 console.log(`Atenção para o horário: ${hora}:${min}:${seg}`)
}
setInterval(mostraHora, 5000)
```

# Funções e Eventos

- ***Funções Anônimas***

- ***Se executarmos o código anterior, será apresentada no console uma nova mensagem a cada 5 segundos***

Atenção para o horário: 11:43:25

Atenção para o horário: 11:43:30

Atenção para o horário: 11:43:35

...



# Funções e Eventos

- ***Funções Anônimas***

- ***Podemos construir esse mesmo script, utilizando uma função anônima***

```
setInterval(() => {
 const data = new Date()
 const hora = data.getHours()
 const min = data.getMinutes()
 const seg = data.getSeconds()
 console.log(`Atenção para o horário: ${hora}:${min}:${seg}`)
}, 5000)
```

- ***Utilizamos uma função anônima como argumento de outras funções***



# Funções e Eventos

- **Funções Anônimas**

- Passamos por **argumentos** uma **função anônima** para a função **setInterval()**
- A **função anônima** passa a ser um **parâmetro** da função **setInterval()**
- A função **setInterval()** executa essa **função anônima** a cada **5 segundos**

# Funções e Eventos

- **Eventos**

- *Um evento é a ocorrência de uma **ação**, geralmente produzida por um **usuário**, em uma página*
- *Clicar em um botão, selecionar um item, sair de um campo, pressionar uma tecla, passar o mouse sobre uma imagem, redimensionar a página são alguns dos eventos que podem ser controlados em um sistema*
- *Adicionar programação nas páginas web associada à ocorrência dos diversos eventos JS permite criar maior interatividade com o usuário, dando maior dinamismo à página*

# Funções e Eventos

- **Eventos**

- *A partir da programação dos eventos, é possível, por exemplo, trocar uma imagem quando o usuário modifica a seleção de um item em uma lista de botões, exibir mensagens de advertência quando o usuário deixa um campo de edição com um conteúdo inválido ou, então, executar uma ação vinculada ao pressionamento de uma determinada tecla, além de vários outros*
- *Podem estar relacionados com **eventos** de **interface** do **usuário** (**load**, **unload**, **resize**), **eventos** de **mouse** (**click**, **dbclick**, **mouseover**), **eventos** de **teclado** (**keypress**, **keydown**, **keyup**) ou **eventos** de **formulário** (**change**, **focus**, **blur**)*

# Funções e Eventos

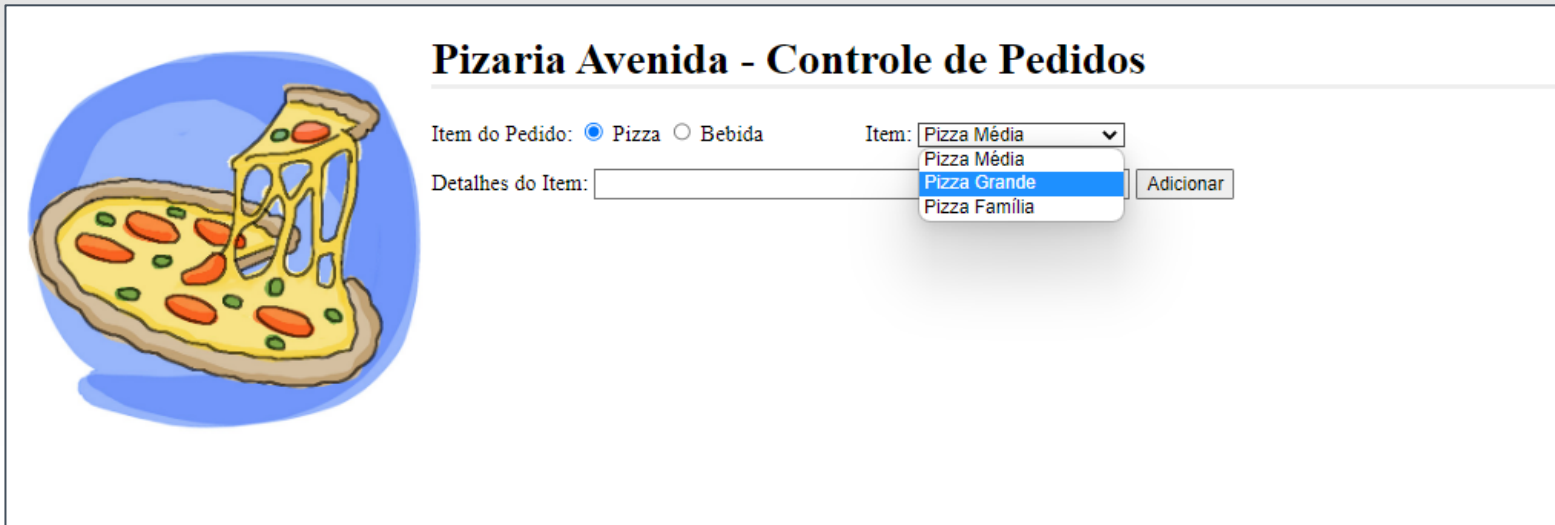
- **Eventos**

- **Exemplo (04):**

- *Explora novos eventos visando dar maior interatividade a um sistema de controle de pedidos de uma pizzeria*
    - *Imagine que a página deve substituir o bloco de pedidos de um garçom, que, utilizará um **tablet** ou **smartphone** para anotar o pedido de cada cliente*
    - *O acesso se dará a partir do navegador do aparelho. A página deve conter recursos que facilitem o atendimento para o garçom*
    - *Um desses recursos é que a lista de itens deve conter apenas pizzas ou apenas bebidas, conforme a seleção inicial do tipo de item*
    - *Quando ocorrer a troca entre pizza e bebidas, o conteúdo da lista de itens do pedido deve ser modificado*

# Funções e Eventos

- **Eventos**
  - **Exemplo (04):**



**Pizzeria Avenida - Controle de Pedidos**

Item do Pedido: ☒ Pizza ☐ Bebida

Item: Pizza Média  
Pizza Média  
Pizza Grande  
Pizza Família

Detalhes do Item:

Adicionar

Figura 5 – Quando ocorre a troca entre Pizza e Bebidas, os itens do campo select são modificados

# Funções e Eventos


- **Eventos**

- **Exemplo (04):**

- *Outro recurso importante para o sistema, que pode ser implementado a partir da programação de eventos JS, é o de **exibir uma dica** quando o usuário posicionar no campo **Detalhes do Item***
    - *A dica deve conter o **número máximo** de sabores da pizza, de acordo com o **tamanho**, selecionado no campo **“Item”***
    - *A mensagem deve ser **exibida** no próprio campo de edição a partir da **propriedade placeholder**, que é um texto apresentado no campo e que **desaparece** quando o usuário inicia a digitação*

# Funções e Eventos

- **Eventos**
  - *Exemplo (04):*



## Pizzaria Avenida - Controle de Pedidos

Item do Pedido: ☒ Pizza ☐ Bebida      Item:

Detalhes do Item:

Pizza Família (Atum, Calabresa, Portuguesa e 4 Queijos)  
Jarra de Suco (Uva)  
Água Mineral (3 copos com gelo)

Figura 6 – A dica é alterada ao posicionar no campo de edição, conforme o tamanho da pizza

# Funções e Eventos

- **Eventos**
  - **Exemplo (04): - Parte 1**

```
1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4 <meta charset="UTF-8">
5 <meta http-equiv="X-UA-Compatible" content="IE=edge">
6 <meta name="viewport" content="width=device-width, initial-scale=1.0">
7 <link rel="stylesheet" href="../css/estilos.css">
8 <title>Exemplo 04 | bkBank Academy</title>
9 </head>
10 <body>
11
12 <h1> Pizaria Avenida - Controle de Pedidos </h1>
13 <form>
14 <p>Item do Pedido:
15 <input type="radio" name="produto" id="rbPizza" checked autofocus> Pizza
16 <input type="radio" name="produto" id="rbBebida"> Bebida
17 Item:
18 <select id="inPizza">
19 <option value="media">Pizza Média</option>
20 <option value="grande">Pizza Grande</option>
21 <option value="familia">Pizza Família</option>
22 </select>
```

Figura 7 – HTML do Exemplo (04) – Parte 1



# Funções e Eventos

- **Eventos**
  - **Exemplo (04): - Parte 2**

```
23 <select id="inBebida" class="oculta">
24 <option value="refri">Refrigerante Litro</option>
25 <option value="suco">Jarra de Suco</option>
26 <option value="agua">Água Mineral</option>
27 </select>
28 </p>
29 <p>Detalhes do Item:
30 <input type="text" id="inDetalhes" class="detalhes">
31 <input type="submit" value="Adicionar">
32 </p>
33 </form>
34 <pre></pre>
35 <script src="../js/exemplo_4.js"></script>
36 </body>
37 </html>
```

Figura 7 – HTML do Exemplo (04) – Parte 2

# Funções e Eventos

- **Eventos**

- **Exemplo (04): - Parte 2**

Criamos dois campos do tipo select: um contendo opções de pizza e outro contendo opções de bebida

Observe que a tag `<select id="inBebida" ...>` contém o atributo `class="oculta"`. E, no arquivo `estilos.css`, está a estilização dessa classe, com a declaração `"display: none;"`

Dessa forma, essa lista não é exibida quando a página é carregada

Figura 7 – HTML do Exemplo (04) – Parte 2

# Funções e Eventos

- **Eventos**
  - **Exemplo (04): - Parte 1**

```
1 const frm = document.querySelector("form") // obtém elementos da página
2 const resp = document.querySelector("pre")
3
4 const itens = [] // vetor global para armazenar os itens do pedido
5
6 frm.rbPizza.addEventListener("click", () => { // quando radio button é clicado
7 frm.inBebida.className = "oculta" // oculta select das bebidas
8 frm.inPizza.className = "exibe" // exhibe select das pizzas
9 })
10
11 frm.rbBebida.addEventListener("click", () => { // quando radio button é clicado
12 frm.inPizza.className = "oculta" // oculta select das pizzas
13 frm.inBebida.className = "exibe" // exhibe select das bebidas
14 })
```

Figura 8 – JS do Exemplo (04) – Parte 1

# Funções e Eventos

- **Eventos**
  - **Exemplo (04): - Parte 2**

```
16 frm.inDetalhes.addEventListener("focus", () => { // quando campo recebe o foco
17 if (frm.rbPizza.checked) { // se radiobutton rbPizza estiver marcado
18 const pizza = frm.inPizza.value // obtém value do item selecionado
19 // uso do operador ternário, para indicar o número de sabores
20 const num = pizza == "media" ? 2 : pizza == "grande" ? 3 : 4
21 // atributo placeholder exibe uma dica de preenchimento do campo
22 frm.inDetalhes.placeholder = `Até ${num} sabores`
23 }
24 })
25
26 frm.inDetalhes.addEventListener("blur", () => { // quando campo perde o foco
27 frm.inDetalhes.placeholder = "" // limpa a dica de preenchimento
28 })
```

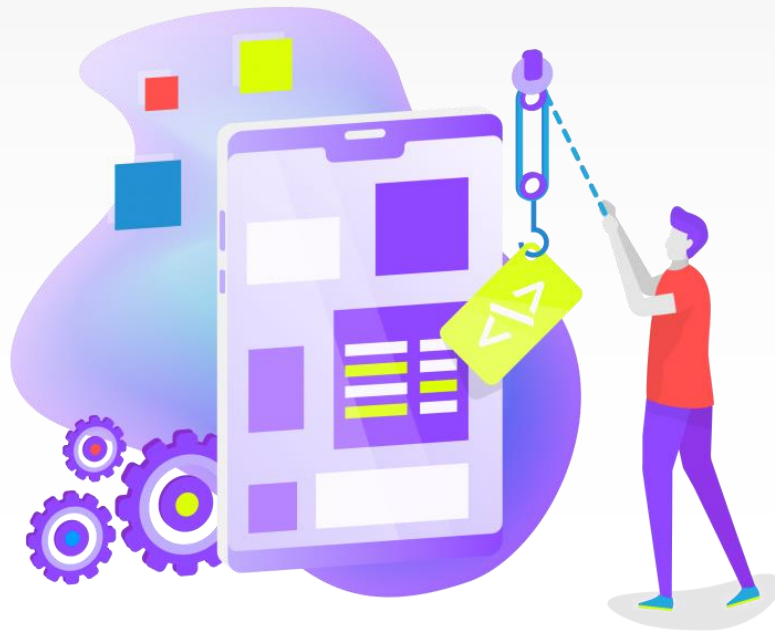
Figura 9 – JS do Exemplo (04) – Parte 2

# Funções e Eventos

- **Eventos**
  - **Exemplo (04): - Parte 3**

```
30 frm.addEventListener("submit", (e) => {
31 e.preventDefault() // evita envio do form
32 let produto
33 if (frm.rbPizza.checked) {
34 const num = frm.inPizza.selectedIndex // obtém nº do item selecionado
35 produto = frm.inPizza.options[num].text // texto do item selecionado
36 } else {
37 const num = frm.inBebida.selectedIndex
38 produto = frm.inBebida.options[num].text
39 }
40 const detalhes = frm.inDetalhes.value // conteúdo do inDetalhes
41 itens.push(produto + " (" + detalhes + ")") // adiciona ao vetor
42 resp.innerHTML = itens.join("\n") // exibe os itens do pedido
43
44 frm.reset() // limpa o form
45 frm.rbPizza.dispatchEvent(new Event("click")) // dispara click em rbPizza
46 })
```

Figura 10 – JS do Exemplo (04) – Parte 3



# EXERCÍCIOS

# Referências

Duckett, J.; Javascript e JQuery - Desenvolvimento de interfaces web interativas. Alta Books, 2018.

Flanagan, D.; JavaScript: The Definitive Guide, 7th Edition. O'Reilly Media, Inc. 2020.

Scott A. D., MacDonald M., Powers S.; JavaScript Cookbook, 3rd Edition. O'Reilly Media, Inc. 2021.

