
UNIPAC
Barbacena

# Centro Universitário Presidente Antônio Carlos
# Programação para Internet

## Funções JavaScript

Felipe Roncalli de Paula Carneiro
felipecarneiro@unipac.br

# O que vamos aprender nessa aula

- Introdução;
- O que é uma Função?
- Qual a imoportância das Funções na programação?
- Objetivo de uma Função
- Componentes de uma Função
- Sintaxe Básica de Funções JavaScript
- Retorno de uma Função
- Funções Anônimas
- Arrow functions
- Importância de nomes de funções descritivos

# O que é uma Função?

# O que é uma Função?

Em programação, uma função é um bloco de código autônomo e reutilizável que executa uma tarefa específica. Ela é projetada para receber um conjunto de entradas (argumentos), executar um conjunto de instruções e, opcionalmente, retornar um resultado. As funções são uma parte fundamental da maioria das linguagens de programação, incluindo JavaScript.

# Qual a imoportância das Funções na programação?

# Qual a imoportância das Funções na programação?

**Reutilização de Código**: As funções permitem que você defina um bloco de código que pode ser executado várias vezes em diferentes partes do seu programa. Isso evita a repetição de código, tornando seu programa mais conciso, mais fácil de manter e menos propenso a erros. Uma vez que você tenha definido uma função, você pode chamá-la quantas vezes quiser, o que economiza tempo e esforço.

# Qual a imoportância das Funções na programação?

**Organização e Modularidade:** À medida que os programas se tornam maiores e mais complexos, fica mais difícil manter tudo em ordem. As funções permitem que você divida o código em partes menores e independentes. Cada função pode ser responsável por uma tarefa específica, facilitando a compreensão do fluxo do programa e a localização de erros.

# Qual a imoportância das Funções na programação?

**Legibilidade:** Funções bem nomeadas podem agir como uma espécie de documentação interna do seu código. Quando você dá um nome significativo a uma função, alguém que lê seu código saberá imediatamente o que ela faz, mesmo sem examinar os detalhes do seu funcionamento interno. Isso torna seu código mais legível e compreensível.

# Qual a imoportância das Funções na programação?

**Manutenção Simplificada:** Se você precisa fazer uma alteração em uma parte específica do seu código, é muito mais fácil quando essa parte está encapsulada em uma função. Você só precisa fazer a alteração na função uma vez, e todas as chamadas para essa função serão afetadas. Isso economiza tempo e reduz a chance de introduzir erros durante a manutenção.

# Qual a imoportância das Funções na programação?

**Isolamento de Escopo:** As funções também ajudam a isolar variáveis dentro de seus próprios escopos. Isso significa que as variáveis definidas dentro de uma função não interferirão nas variáveis definidas fora dela. Isso é especialmente útil para evitar conflitos de nomes de variáveis e para garantir que as alterações locais não afetem outras partes do código.

# Qual a imoportância das Funções na programação?

**Abstração:** As funções permitem abstrair a complexidade. Você pode encapsular uma série de operações complexas em uma única função e chamá-la quando necessário, em vez de lidar com esses detalhes complexos sempre que precisar realizar a ação.

# Qual a imoportância das Funções na programação?

**Criação de APIs:** Em um contexto mais amplo, funções bem projetadas podem formar APIs (Interfaces de Programação de Aplicativos) que podem ser usadas por outros desenvolvedores para interagir com seu código de maneira padronizada. Isso é especialmente importante quando se cria bibliotecas ou frameworks reutilizáveis.

# Qual a imoportância das Funções na programação?

**Testabilidade:** Funções independentes podem ser testadas individualmente, o que facilita a detecção de erros e a realização de testes de unidade.
Isso contribui para a criação de código mais robusto e confiável.

# Qual a imoportância das Funções na programação?

**Colaboração em Equipe:** O uso de funções facilita a divisão do trabalho em equipes, pois cada membro pode trabalhar em funções separadas sem interferir no trabalho dos outros.

# Qual a imoportância das Funções na programação?

Em resumo, as funções são blocos de construção fundamentais na programação. Elas oferecem reutilização de código, organização, legibilidade e facilitam a manutenção. O uso eficaz de funções torna o processo de desenvolvimento mais eficiente e ajuda a criar programas mais robustos e escaláveis.

# Objetivo de uma Função

As funções servem para encapsular lógica e operações em unidades gerenciáveis e organizadas. Elas oferecem várias vantagens, como modularidade, reutilização de código e legibilidade.

# Componentes de uma Função

**Declaração/Definição da Função:**

- A declaração de uma função envolve atribuir um nome único a ela, para que possa ser referenciada mais tarde no código.
- Em JavaScript, você pode declarar uma função usando a palavra-chave function.

# Componentes de uma Função

**Parâmetros:**

- Parâmetros são valores que a função espera receber quando é chamada. Eles atuam como entradas para a função.
- As funções podem ter zero ou mais parâmetros, que são listados entre parênteses na declaração da função.

# Componentes de uma Função

**Corpo da Função:**

- O corpo da função é o bloco de código que contém as instruções a serem executadas quando a função é chamada.
- Essas instruções podem envolver cálculos, manipulação de dados, controle de fluxo (como condicionais e loops) e muito mais.

# Componentes de uma Função

**Retorno:**

- Algumas funções retornam um valor após a execução. O valor retornado é especificado usando a palavra-chave **return**.
- O retorno permite que a função forneça um resultado que pode ser usado em outros lugares do programa.

# Componentes de uma Função

**Chamada da Função:**

- Para executar uma função, você a chama pelo nome e fornece os argumentos esperados.
- A chamada da função pode ser feita em qualquer parte do código onde a função seja visível.

# Componentes de uma Função

```
1  // Declaração da função
2  function saudacao(nome) {
3      return "Olá, " + nome + "!";
4  }
5
6  // Chamada da função
7  var mensagem = saudacao("Alice");
8  console.log(mensagem); // Saída: "Olá, Alice!"
```

# Componentes de uma Função

```
1  // Declaração da função
2  function saudacao(nome) {
3      console.log("Olá, " + nome + "!");
4  }
5
6  // Chamada da função
7  saudacao("Alice");
```

# Sintaxe Básica de Funções JavaScript

```javascript
function nomeDaFuncao(parametro1, parametro2, /* ... */) {
    // Corpo da função: conjunto de instruções a serem executadas
    // Pode incluir cálculos, manipulação de dados, controle de fluxo, etc.

    // Opcional: retornar um valor usando a palavra-chave 'return'
    return resultado;
}
```

# Sintaxe Básica de Funções JavaScript

**Chamada da Função:**

- Para executar uma função, você a chama pelo nome e fornece os argumentos esperados.
- A chamada da função pode ser feita em qualquer parte do código onde a função seja visível.

# Sintaxe Básica de Funções JavaScript

**function:** A palavra-chave que indica a definição de uma função.

**nomeDaFuncao:** O nome exclusivo dado à função. Esse nome deve seguir as regras de nomenclatura de variáveis em JavaScript.

**Parâmetros:** Uma lista de variáveis separadas por vírgulas, definidas entre parênteses (). Esses parâmetros representam os valores que você pode passar para a função quando ela é chamada.

# Sintaxe Básica de Funções JavaScript

**Corpo da Função:** O bloco de código entre chaves {} que contém as instruções a serem executadas quando a função é chamada.

**return:** A palavra-chave opcional usada para retornar um valor da função. O valor pode ser qualquer tipo de dado (número, string, objeto, etc.).

# Retorno de uma Função

O retorno de valores em funções é uma parte fundamental da programação, pois permite que uma função não apenas execute um conjunto de instruções, mas também forneça um resultado que pode ser utilizado em outras partes do código. Vamos entender melhor como funciona o retorno de valores em funções:

# Retorno de uma Função

**Retorno de Valores**: Quando uma função é definida para retornar um valor, ela pode usar a palavra-chave return para especificar qual valor deve ser devolvido quando a função é chamada.

**Como Funciona**: Quando a execução da função atinge uma declaração return, a função para de ser executada e o valor especificado é imediatamente retornado à parte do código que chamou a função.

**Utilização**: O valor retornado pela função pode ser atribuído a uma variável ou usado de qualquer outra forma necessária.

# Funções Anônimas

**Funções Anônimas:** Uma função anônima é uma função que não possui um nome identificador associado a ela. Elas são frequentemente usadas quando você precisa passar uma função como argumento para outra função (como em callbacks) ou quando deseja definir uma função como uma expressão. Aqui está um exemplo de uma função anônima

# Funções Anônimas

```
1  var saudacao = function(nome) {
2      return "Olá, " + nome + "!";
3  };
4
5  console.log(saudacao("Alice"));
6  // Saída: "Olá, Alice!"
7
```

Neste exemplo, a função saudacao é definida sem um nome, mas é atribuída a uma variável. Você pode chamá-la usando o nome da variável, como faria com qualquer outra função.

# Funções Anônimas

**Funções Anônimas:** Uma função anônima é uma função que não possui um nome identificador associado a ela. Elas são frequentemente usadas quando você precisa passar uma função como argumento para outra função (como em callbacks) ou quando deseja definir uma função como uma expressão. Aqui está um exemplo de uma função anônima

# Funções Anônimas

**Callbacks:** Funções anônimas são frequentemente usadas como callbacks, que são funções passadas como argumentos para outras funções. Isso é especialmente comum em eventos assíncronos, como requisições HTTP. Veja um exemplo

```
// Função anônima como callback
fetch('https://api.example.com/data')
    .then(response => response.json())
    .then(data => console.log(data))
    .catch(error => console.error(error));
```

# Funções Anônimas

**Expressões de Função:** As funções anônimas também podem ser usadas como expressões em locais onde você normalmente usaria um valor ou variável. Por exemplo:

```
var saudacao = function(nome) {
    return "Olá, " + nome + "!";
};

console.log(saudacao("Alice"));
```

# Funções Anônimas

**Funções de IIFE (Immediately Invoked Function Expression):** As funções anônimas também podem ser invocadas imediatamente após serem definidas. Isso é útil para criar escopos isolados e evitar poluição do escopo global.

```
(function() {
    var mensagem = "Esta é uma função IIFE.";
    console.log(mensagem);
})();
```

# Funções Anônimas - Vantagens e Utilidades

- **Flexibilidade:** Funções anônimas são altamente flexíveis e podem ser definidas em qualquer contexto onde você precisa de uma função.
- **Callbacks:** São comumente usadas como callbacks em operações assíncronas para lidar com resultados quando estiverem prontos.
- **Encapsulamento:** Permitem encapsular lógica em unidades isoladas, mantendo o escopo local.

# Funções Anônimas - Vantagens e Utilidades

- **Funções de Primeira Classe:** Podem ser atribuídas a variáveis, armazenadas em arrays e objetos, e passadas como argumentos para outras funções.
- **Legibilidade e Organização:** Quando usado de forma adequada, o uso de funções anônimas pode melhorar a legibilidade e a organização do código.

# Funções Anônimas

Funções anônimas são uma ferramenta poderosa em JavaScript que oferece flexibilidade, encapsulamento e a capacidade de criar estruturas modulares e reutilizáveis em seu código. Elas são especialmente úteis para trabalhar com eventos assíncronos, expressões de função e para criar funções IIFE para criar escopos isolados. Dominar o uso de funções anônimas amplia suas habilidades de programação e ajuda a criar código mais eficiente e organizado.

# Arrow functions

**Arrow Functions:** As arrow functions são uma adição poderosa ao JavaScript que oferece uma sintaxe mais concisa para a criação de funções. Elas tornam o código mais enxuto e expressivo, especialmente para funções curtas. Nesta seção, exploraremos o que são as arrow functions e como elas podem simplificar a criação de funções em JavaScript.

# Arrow functions - Sintaxe Básica

A sintaxe das arrow functions inclui a notação da seta =>, que segue os parâmetros da função e precede o corpo da função.

```
const funcaoArrow = (parametro1, parametro2) => {
    // Corpo da função
    return resultado;
};
```

# Arrow functions - Vantagens

**Sintaxe Concisa**: A sintaxe encurtada torna as arrow functions ideais para funções curtas e expressões simples.

**Redução de Código**: Arrow functions eliminam a necessidade de usar a palavra-chave function, tornando o código mais limpo.

**Clareza**: Em muitos casos, as arrow functions tornam o código mais claro, reduzindo a quantidade de caracteres e detalhes desnecessários.

# Arrow functions - Vantagens

As arrow functions são uma adição poderosa à sintaxe do JavaScript, oferecendo uma maneira mais concisa de criar funções. Elas são especialmente úteis para funções curtas e anônimas, onde a legibilidade e a eficiência são fundamentais. Ao dominar as arrow functions, você pode tornar seu código mais eficiente e expressivo.

# Funções anônimas e arrow functions

```
1  // Função anônima tradicional
2  var soma = function(a, b) {
3      return a + b;
4  };
5
6  // Arrow function equivalente
7  const somaArrow = (a, b) => a + b;
8
```

# Importância de nomes de funções descritivos

A importância de usar nomes de funções descritivos não pode ser subestimada na programação. O nome de uma função é a primeira pista que os leitores do seu código têm sobre o que a função faz e como ela é usada. Nomes de funções claros e descritivos melhoram a legibilidade, compreensão e manutenção do código. Aqui estão alguns motivos pelos quais nomes de funções descritivos são cruciais:

# Importância de nomes de funções descritivos

```
// Nome não descritivo
function f() {
    // ...
}
// Nome descritivo
function calcularAreaRetangulo(largura, altura) {
    // ...
}
// Nome não descritivo
const x = function(y) {
    // ...
};
// Nome descritivo
const validarEmail = function(email) {
    // ...
};
```

# Exercícios

# Dúvidas??