MATERIAL EXTRA

Funções

growdev

FUNÇÕES

Uma função é um procedimento de JavaScript - um conjunto de instruções que executa uma tarefa. De modo geral, função é um "subprograma" que pode ser chamado por código externo (ou interno no caso de recursão) à função.

Assim como o programa em si, uma função é composta por uma sequência de instruções chamada corpo da função. Valores podem ser passados para uma função e ela pode ou não retornar um valor.

O jeito mais básico de definir funções em JavaScript é através da **function declaration**, onde toda função de declaração começa com a palavra reservada `function`, seguida pelo nome da função (obrigatório) e uma lista de parâmetros (opcionais) separados por vírgula e encapsulados em parênteses (obrigatórios), o último passo é definir as chaves (obrigatórias) que será o corpo da função.

Estrutura de uma função simples:

Exemplo de definição de uma função:

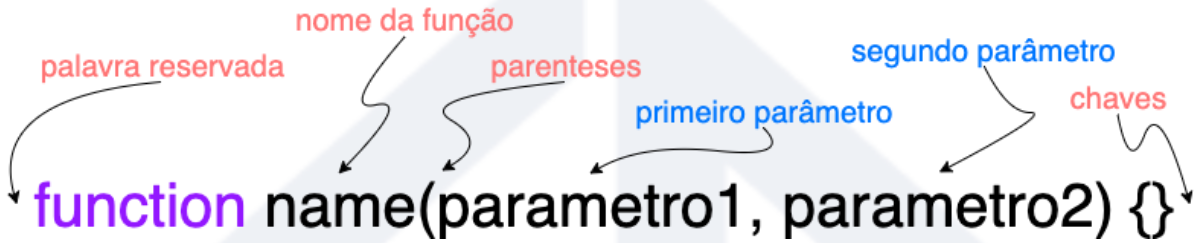
```
function saudacao() {
  console.log('Olá, mundo!')
}

// Chamada da função
saudacao()
```

Parâmetros e Argumentos

As funções podem receber parâmetros, que são valores passados quando a função é chamada. Esses valores podem ser usados dentro da função para realizar operações.

Estrutura de uma função com parâmetros:



Exemplo de definição de uma função com parâmetros:

```
function saudacao(nome) {
  console.log(`Olá, ${nome}!`)
}

// Chamada da função
saudacao("Growdever")
```

Quando chamamos a função e passamos o nome como argumento. No exemplo acima, "**Growdever**" é o argumento que substitui o parâmetro `nome` na função.

Retorno de Funções

As funções podem ou não retornar informações, nos exemplos acima as funções efetuavam ações e não retornavam nada. Nos casos em que a função retorne informações precisamos utilizar a palavra reservada **`return`** para definir o valor que será retornado para quem chamou a função.

```
function soma(num1, num2) {
  return num1 + num2;
}

let resultado = soma(5, 3); // resultado será 8
console.log(resultado); // Saída: 8
```

A função `soma` recebe dois números como parâmetros e retorna a soma deles.

Funções anônimas

Funções anônimas em JavaScript são funções que não possuem um nome explícito na sua declaração. Ao contrário das funções nomeadas, essas funções são usadas principalmente em contextos onde a função é passada como argumento, atribuída a uma variável, ou definida diretamente onde é usada. Elas são uma ferramenta poderosa para a criação de código mais conciso e dinâmico, especialmente em programação funcional e na manipulação de eventos.

Definição de funções anônimas

Uma função anônima é declarada de maneira semelhante a uma função tradicional, exceto que ela não possui um nome. Geralmente, essas funções são atribuídas a uma variável ou usadas diretamente em um contexto onde não há necessidade de chamá-las repetidamente com um nome específico.

Exemplo básico de função anônima atribuída a uma variável:

```
let saudacao = function (nome) {
  console.log(`Olá, ${nome}`)
}

saudacao('Growdever') // Saída: Olá, Growdever!
```

Neste exemplo, a função anônima está sendo atribuída à variável `saudacao`. A partir daí, podemos chamar a função usando a variável como se fosse uma função nomeada.

Arrow Function

A introdução das arrow functions em JavaScript (ES6) oferece uma maneira mais curta e concisa de declarar funções anônimas e são frequentemente preferidas em callbacks por serem menos verbosas.

Exemplo básico de função anônima atribuída a uma variável:

```
let saudacao = (nome) => {
  console.log(`Olá, ${nome}`)
}

saudacao('Growdever') // Saída: Olá, Growdever!
```

Da mesma forma que no primeiro exemplo da função anônima, a arrow function acima está sendo atribuída à variável `saudacao`. A partir daí, podemos chamar a função usando a variável como se fosse uma função nomeada.

Contextos em que Funções Anônimas são Usadas

Funções anônimas são amplamente utilizadas em diversos contextos em JavaScript, tais como:

1. Funções de Callback

Um dos usos mais comuns de funções anônimas é como callback. Uma função de callback é uma função passada como argumento para outra função, e é executada dentro dessa função em um momento posterior.

Exemplo com uma função de callback:

```
setTimeout(function () {
  console.log('Isto aparece após 2 segundos')
}, 2000)
```

```
setTimeout(() ⇒ {
  console.log('Isto aparece após 2 segundos')
}, 2000)
```

Aqui, as funções anônimas são passadas como argumento para setTimeout, que as executa após 2 segundos.

2. Funções Imediatamente Invocadas (IIFE)

Funções imediatamente invocadas (*Immediately Invoked Function Expressions* - IIFE) são funções anônimas que são executadas imediatamente após a sua criação. Elas são usadas frequentemente para criar escopos isolados e evitar poluição do escopo global.

Exemplo de IIFE:

```
(function () {
 console.log('Esta função é executada imediatamente!')
})()
```

```
(() ⇒ {
  console.log('Esta função é executada imediatamente!')
})()
```

Nos exemplos acima, as funções anônimas são invocadas imediatamente após suas declarações, sem a necessidade de serem chamadas separadamente.

3. Manipulação de Eventos

Funções anônimas são muito usadas em eventos, como cliques de botões, onde uma função específica não precisa ser reutilizada e pode ser criada diretamente no local onde o evento é tratado.

Exemplo de uso com eventos:

```
document.addEventListener('mouseleave', function () {
  console.log('O mouse saiu do documento HTML!')
})
```

```
document.addEventListener('mouseleave', () ⇒ {
  console.log('O mouse saiu do documento HTML!')
})
```

Nestes casos, as funções anônimas são chamadas toda vez que o usuário tirar o cursor do mouse do documento HTML, mas não precisam ter um nome já que não serão chamadas fora desse contexto.

Vantagens das Funções Anônimas

1. **Concisão**: Funções anônimas podem ser declaradas de maneira rápida e utilizada imediatamente, sem a necessidade de nomeá-las, o que deixa o código mais curto e direto.

2. **Escopo Local**: Como elas são frequentemente utilizadas em expressões ou em escopos locais, ajudam a evitar a poluição do escopo global, o que pode melhorar a manutenção do código em projetos grandes.

3. **Uso Temporário**: São úteis para criar funções que só serão usadas uma vez, como em eventos ou callbacks, onde não faz sentido dar um nome a algo que não será reutilizado.

Desvantagens das Funções Anônimas

Apesar de suas vantagens, o uso de funções anônimas também apresenta alguns desafios:

1. **Depuração mais difícil**: Quando uma função anônima gera um erro, a stack trace (rastreamento de erro) pode ser menos útil, já que o erro será atribuído a uma função sem nome. Isso pode tornar a depuração mais complicada, especialmente em projetos grandes.

2. **Legibilidade**: Em alguns casos, o uso excessivo de funções anônimas pode tornar o código menos legível. Funções nomeadas frequentemente tornam o propósito e a intenção mais claros, enquanto funções anônimas podem parecer criptográficas para outros desenvolvedores que estão revisando o código.

Funções Anônimas vs Funções Nomeadas

Característica	Função Anônima	Função Nomeada
Nome	Não possui nome	Possui um nome definido
Reutilização	Não pode ser reutilizada fora do contexto imediato	Pode ser chamada em qualquer lugar no escopo
Legibilidade	Pode ser mais difícil de entender em certos contextos	Torna o propósito da função mais claro
Uso em callbacks e eventos	Muito comum	Menos comum, mas pode ser usada
Stack trace	Difícil de depurar em caso de erro	Mais fácil de identificar problemas

As funções em JavaScript são um dos pilares da linguagem, permitindo a criação de código organizado, reutilizável e mais eficiente. Ao entender como criar, utilizar e retornar valores em funções, o desenvolvedor ganha flexibilidade e poder para construir soluções mais robustas.

BONS ESTUDOS

growdev