Apresentação da Disciplina

Dia 6
Funções - Parte 1

Centro de Alta Performance - SECITECI

SECITECI
SECRETARIA DE ESTADO DE CIÊNCIA E TECNOLOGIA E INOVAÇÃO

GOVERNO DE MATO GROSSO

<Fic_Dev>
Programador de Sistemas

Roteiro

1. Desafio
2. Objetivo da disciplina
3. Pré-requisitos
4. Conteúdo

Desafio

Crie um programa em JavaScript que tenha uma função chamada "**calcularAreaRetangalo**", ela deve receber os parâmetros base e altura, e deve retornar a área do retângulo. Lembre-se de fazer o tratamento das exceções.

Objetivo da disciplina

Compreender a definição e a utilidade das funções no JavaScript.

Exercitar a criação de funções.

Compreender e exercitar o tratamento de exceções

Pré-requisitos

- Sólidos conhecimentos em Lógica
- Alto poder de abstração
- Muita Força de Vontade, Persistência e Atenção
- Ser autodidata ;-)

<Fic_Dev>
Programador de Sistemas

Conteúdo

Conteúdo

Uma função em JavaScript é um **bloco de código** que executa uma tarefa específica.

As funções são usadas para **modularizar** o código, tornando-o mais fácil de ler e manter.

As funções também são usadas para **reutilizar** código, permitindo que você escreva uma vez e use várias vezes em seu programa.

Conteúdo

- **Sintaxe:**

```
function nome_da_funcao(parametro1, parametro2, ..., parametroN) {
    // Bloco de código que executa a tarefa da função
}
```

Conteúdo

- **Exemplo:**
  Vamos criar uma função simples que imprima uma frase no console.

```
function diga_ola() {
    console.log("Olá");
}
// Chamando uma função:
diga_ola(); // "Olá"
```

Conteúdo

- **Parâmetros de funções**

Conteúdo

As funções em JavaScript podem receber parâmetros para processá-los
Os parâmetros são passados para a função como argumentos quando a função é chamada.
Os parâmetros são opcionais e você pode passar quantos parâmetros desejar.

```
function somar(valor1, valor2) {
    console.log(valor1 + valor2);
}
// Chamando uma função:
somar(10, 2); // 12
```

<Fic_Dev>
Programador de Sistemas

Conteúdo

- **Retorno de funções**

Conteúdo

As funções em JavaScript podem retornar valores usando a palavra-chave *return*.

Quando uma função encontra a palavra-chave *return*, ela interrompe a execução e retorna um valor para quem chamou a função.

Exemplos:

Conteúdo

```
function somar(valor1, valor2) {
    return valor1 + valor2;
}
// Chamando uma função:
var resultado = somar(5, 2);
console.log(resultado) // 7
```

Conteúdo

Considere a seguinte função que verifica se um número é par ou ímpar:

```
function par_ou_impar(numero) {
    if (numero % 2 === 0) {
        return "par";
    }
    return "ímpar";
}
// Chamando a função
var resultado = par_ou_impar(2);
console.log(resultado); // "par"
```

Conteúdo

- **Exceções**

Conteúdo

No JavaScript, podemos **tratar exceções** que ocorrem durante a execução de um programa.
O bloco **try-catch** é uma construção que permite capturar e lidar com essas exceções de forma estruturada.

```
try {
    // código que pode gerar exceções
} catch (erro) {
    // tratamento da exceção
}
```

Conteúdo

- **Exemplos:**

```
try {
    // Código que pode lançar uma exceção
    console.log('Executando o código do bloco try...');
    throw new Error('Ocorreu um erro¡);
} catch (excecao) {
    // Tratamento de exceção
    console.log('Executando o código do bloco catch.');
    console.log(excecao.message); // Ocorreu um erro!
}
```

<Fic_Dev>
Programador de Sistemas

Conteúdo

Podemos também criar funções que retornam exceções.

Vamos ver um exemplo:

Conteúdo

```
function dividir(a, b) {
    if (b === 0) {
        throw new Error("Divisão por zero não é permitida.");
    }
    return a / b;
}
// Tratando a exceção
try {
    // Chamando a função
    const resultado = dividir(10, 0);
    console.log('O resultado é:', resultado);
} catch (error) {
    console.log('Ocorreu um erro:', error.message);
}
```

Conteúdo

Além disso, é possível utilizar a cláusula **finally** para definir um bloco de código que será executado independentemente de ocorrer ou não uma exceção.

Este bloco é opcional e pode ser utilizado para limpar recursos.

Conteúdo

```
try {
    // código que pode gerar exceções
} catch (e) {
    // tratamento da exceção
} finally {
    // código que será executado com ou sem exceção
}
```

Conteúdo

Em resumo, a utilização dos blocos **try**, **catch** e **finally** é importante para garantir que exceções sejam tratados de forma adequada durante a execução de um programa em JavaScript.

Questionamentos?

Muito Obrigado !

<Fic_Dev>
Programador de Sistemas