**Callback**

Um callback é uma função passada como argumento para outra função, executada após a conclusão de uma operação assíncrona ou de um evento específico. Sua principal finalidade é permitir a execução assíncrona de código, sem interromper a execução do restante do programa. Exemplos comuns incluem operações como solicitações de rede, manipulação de arquivos ou temporizadores.

// Função que simula uma operação assíncrona com um atraso de 2 segundos.

//Após 2 segundos, chamar o callback

function operacaoAssincrona(callback) {

setTimeout(function() {

callback("Operação assíncrona concluída!");

}, 2000);

}

function meuCallback(mensagem) {

console.log("Callback recebido:", mensagem);

}

console.log("Iniciando operação assíncrona...");

operacaoAssincrona(meuCallback);

console.log("Operação assíncrona iniciada! Aguardando o callback...");

**Promisse**

Uma Promise é um objeto em JavaScript que representa o resultado de uma operação assíncrona que pode estar pendente, resolvida ou rejeitada. Promises são amplamente utilizadas para lidar com operações assíncronas de uma maneira mais elegante e eficiente do que os callbacks tradicionais.

Um exemplo de Promise em JavaScript é uma operação de solicitação HTTP usando a função fetch para obter dados de uma API. Aqui está um exemplo:

// Função que realiza uma solicitação HTTP usando fetch, retorna uma Promise que resolve quando a solicitação HTTP é bem-sucedida

function obterDadosDaAPI() {

return fetch('https://api.example.com/data')

.then(function(response) {

if (!response.ok) {

throw new Error('Erro ao obter dados da API');

}

return response.json();

})

.catch(function(error) {

console.error('Erro na solicitação HTTP:', error);

throw error;

});

}

// Exemplo de uso da Promise

console.log('Iniciando solicitação HTTP...');

obterDadosDaAPI()

.then(function(data) {

console.log('Dados obtidos:', data);

})

.catch(function(error) {

console.error('Erro ao obter dados:', error);

});

**Função anônima**

Uma função anônima é uma função que não possui um nome associado a ela. Elas são definidas inline, diretamente onde são necessárias, e geralmente são passadas como argumentos para outras funções ou atribuídas a variáveis.

setTimeout(function() {

console.log('Esta é uma função anônima sendo executada após 2 segundos.');

}, 2000)

**Ternário**

O operador ternário é uma expressão condicional que permite avaliar uma expressão e retornar um valor com base em uma condição. Ele é frequentemente utilizado como uma forma mais concisa de expressar estruturas condicionais simples.

var idade = 18;

var status = (idade >= 18) ? 'adulto' : 'menor';

console.log(status);

**try/catch**

O bloco try/catch é usado em JavaScript para lidar com exceções (erros) que podem ocorrer durante a execução do código. Ele permite que você envolva o código suscetível a erros dentro de um bloco try e capture qualquer exceção que ocorra dentro desse bloco usando o bloco catch.

try {

// Código que pode gerar uma exceção

var resultado = dividir(10, 0);

console.log("O resultado da divisão é:", resultado);

} catch (erro) {

console.error("Ocorreu um erro:", erro.message);

}

function dividir(dividendo, divisor) {

if (divisor === 0) {

throw new Error("Não é possível dividir por zero.");

}

return dividendo / divisor;

}

**Classe**

Em programação orientada a objetos, uma classe é um modelo ou uma estrutura para criar objetos. Ela define os atributos e comportamentos que os objetos de uma classe específica terão. Os objetos são instâncias de uma classe, o que significa que eles herdam as propriedades e métodos definidos na classe.

class Pessoa {

constructor(nome, idade) {

this.nome = nome;

this.idade = idade;

}

exibirDetalhes() {

console.log(`Nome: ${this.nome}, Idade: ${this.idade}`);

}

}

var pessoa1 = new Pessoa("João", 30);

pessoa1.exibirDetalhes(); // Saída: Nome: João, Idade: 30

**Método**

Em programação orientada a objetos, um método é uma função associada a uma classe ou a um objeto. Ele define o comportamento ou as ações que um objeto pode realizar. Métodos são parte integrante das classes e são usados para manipular os dados dentro dos objetos ou para interagir com outros objetos.

class Circulo {

constructor(raio) {

this.raio = raio;

}

calcularArea() {

return Math.PI \* this.raio \*\* 2;

}

}

var circulo1 = new Circulo(5);

var area = circulo1.calcularArea();

console.log("Área do círculo:", area);

**Objeto/instância de classe**

Um objeto, ou instância de classe, é uma entidade concreta criada a partir de uma classe na programação orientada a objetos. Uma classe atua como um modelo para criar objetos com características e comportamentos específicos. Ao criar um objeto a partir de uma classe, está-se instalando essa classe, gerando uma instância única. Cada objeto criado pode ter seus próprios valores para os atributos da classe e pode invocar os métodos definidos na classe.