Linguagens de Script para Web

Estruturas de Controle e Repetição
Arrays
Trabalhando com Funções
Introdução à Orientação a Objetos

Centro Universitário Senac

Curso: Tecnologias em Sistemas para Internet
Professor: Dennis Lopes da Silva

Estruturas de Controle e Repetição

O que são?

As estruturas de controle nos permitem executar blocos de código condicionalmente, ou seja, somente se uma condição for verdadeira.

if/else

O **if** executa um bloco de código se a condição for verdadeira. O **else** oferece uma alternativa se a condição for falsa.

if/else

```
let idade = 18;

if (idade >= 18) {
  console.log("Você pode dirigir.");
} else {
  console.log("Você ainda não pode dirigir.");
}
```

else if

Use **else if** para testar múltiplas condições em sequência.

else if

```
let nota = 75;

if (nota >= 90) {
  console.log("Excelente!");
} else if (nota >= 70) {
  console.log("Bom!");
} else {
  console.log("Precisa melhorar.");
}
```

switch

Use o **switch** quando você tem uma única expressão e precisa compará-la com vários valores diferentes.

switch

```
let statusDoPedido = "entregue";
let mensagem;

switch (statusDoPedido) {
  case "pendente":
    mensagem = "Seu pedido está sendo preparado.";
    break;
  case "em transporte":
    mensagem = "Seu pedido está a caminho!";
    break;
  case "entregue":
    mensagem = "Seu pedido foi entregue com sucesso!";
    break;
  default:
    mensagem = "Status do pedido desconhecido.";
    break;
}
console.log(mensagem);
```

Loops

Os loops executam um bloco de código várias vezes. Eles são essenciais para processar listas de dados ou realizar tarefas repetitivas.

for

for: Use for quando você sabe exatamente quantas vezes a repetição deve ocorrer.

for

```
for (let i = 0; i < 5; i++) {
  console.log("Contagem: " + i);
}
```

while

while: Use while quando a condição de parada depende de algo que pode mudar dentro do loop, e você não sabe o número exato de repetições.

while

```
let contador = 0;
while (contador < 3) {
  console.log("Contador: " + contador);
  contador++;
}
```

do ... while

É parecido com o **while**, mas a diferença é que o bloco de código é executado pelo menos uma vez, antes que a condição seja testada.

do ... while

```
let numero = 5;
do {
  console.log("O número é " + numero);
  numero++;
} while (numero < 5);
```

Arrays

O que são Arrays?

Arrays são listas ordenadas de valores.

Eles podem armazenar qualquer tipo de dado: números, strings, objetos, e até mesmo outros arrays.

Em JavaScript, os arrays são indexados por zero, o que significa que o primeiro elemento está no índice 0, o segundo no índice 1, e assim por diante.

Arrays

Criando um array:

```
// Array de números
let numeros = [1, 2, 3, 4, 5];

// Array de strings
let frutas = ["Maçã", "Banana", "Morango"];

// Array misto (não é uma boa prática, mas é possível)
let misto = ["Olá", 10, true];
```

Arrays

Acessando elementos:

```
let primeiroNumero = numeros[0]; // Retorna: 1
let segundaFruta = frutas[1];   // Retorna: "Banana"
```

Arrays

Alguns métodos comuns:

.length: Retorna o número de elementos no array.

Exemplo: frutas.**length**;

.indexOf(): Encontra o índice de um elemento.

Exemplo: frutas.**indexOf**(**"Banana"**);

Arrays – Métodos Úteis

.push

.pop

.shift

.unshift

.map

.filter

.reduce

.forEach

.find

.join

Array.push()

Adiciona um ou mais elementos ao final do array.

Exemplo:

```
const arr = [1, 2];
arr.push(3);
console.log(arr); // [1, 2, 3]
```

Array.pop()

Remove o último elemento do array e retorna ele.

Exemplo:

```
const arr = [1, 2, 3];
const removido = arr.pop();
console.log(removido); // 3
console.log(arr); // [1, 2]
```

Array.shift()

Remove o primeiro elemento do array e retorna ele.

Exemplo:

```
const arr = [1, 2, 3];
const removido = arr.shift();
console.log(removido); // 1
console.log(arr); // [2, 3]
```

Array.unshift()

Adiciona um ou mais elementos no início do array.

Exemplo:

```
const arr = [2, 3];
arr.unshift(1);
console.log(arr); // [1, 2, 3]
```

Array.map()

Cria um novo array aplicando uma função a cada elemento do array original. Não modifica o array original.

Exemplo:

```
const numeros = [1, 2, 3];
const dobrados = numeros.map(n => n * 2);
console.log(dobrados); // [2, 4, 6]
```

Array.filter()

Cria um novo array contendo apenas os elementos que passam no teste definido por uma função.

Exemplo:

```
const numeros = [1, 2, 3, 4];
const maioresQueDois = numeros.filter(n => n > 2);
console.log(maioresQueDois); // [3, 4]
```

Array.reduce()

Executa uma função acumuladora em todos os elementos do array, retornando um único valor.

Exemplo:

```
const arr = [1, 2, 3, 4];
const soma = arr.reduce((acumulador, valor) => acumulador + valor, 0);
console.log(soma); // 10
```

Array.forEach()

Executa uma função para cada elemento do array (não retorna nada).

Exemplo:

```
const arr = [1, 2, 3];
arr.forEach(n => console.log(n * 2));
// Imprime 2, 4, 6 no console
```

Array.find()

Retorna o primeiro elemento que satisfaz a condição da função.

Exemplo:

```
const arr = [1, 2, 3, 4];
const encontrado = arr.find(n => n > 2);
console.log(encontrado); // 3
```

Array.join()

O método **join()** une todos os elementos de um array em uma única string.

Você pode passar um argumento opcional (um separador) para especificar o que deve ser colocado entre cada elemento da string resultante.

Se nenhum separador for fornecido, os elementos são separados por vírgula.

Array.join()

Exemplo:

```
let palavras = ["Olá", "mundo", "do", "JavaScript"];
// Juntando os elementos com um espaço como separador
const frase = palavras.join(" ");
// Saída: ["Olá", "mundo", "do", "JavaScript"]
console.log("Array original: ", palavras);
// Saída: "Olá mundo do JavaScript“
console.log("String resultante: ", frase);

// Exemplo com um separador diferente
const listaComHifens = palavras.join(" - ");
// Saída: "Olá - mundo - do - JavaScript"
console.log("String com hífens: ", listaComHifens);
```

Funções

As funções são um dos blocos de construção mais fundamentais de JavaScript.

Em sua essência, uma função é um bloco de código projetado para executar uma tarefa específica.

Pense nelas como pequenos programas que você pode escrever e reutilizar quantas vezes quiser.

Funções

Vantagens

- Reutilização do código
- Organização
- Abstração

Funções

Parâmetros: São os valores que uma função espera receber quando é chamada. Eles são listados entre os parênteses na definição da função**.**

Retorno: A palavra-chave **return** é usada para enviar um valor de volta para o local onde a função foi chamada. Se uma função não tiver um **return**, ela retorna **undefined** por padrão**.**

Funções

Exemplo 1:

```
function saudacao(nome) {
  console.log("Olá, " + nome + "!");
}

saudacao("João"); // Saída: Olá, João!
```

Funções

Exemplo 2:

```
function calcularArea(largura, altura) {
  return largura * altura;
};

let area = calcularArea(10, 5);
console.log(area); // Saída: 50
```

Funções

Exemplo 3 (Função de Seta):

```
// Função com um único parâmetro e uma única linha de código
const dobrar = numero => numero * 2;

console.log(dobrar(7)); // Saída: 14

// Função com múltiplos parâmetros
const somar = (a, b) => a + b;

console.log(somar(10, 20)); // Saída: 30
```

Interação com Usuário

Exemplo:

```
let nome = prompt("Qual seu nome?");
alert("Olá, " + nome + "!");
```

Orientação a Objetos

Paradigma de programação que organiza o código em torno de "**objetos**".

Objetos contêm dados (atributos) e comportamentos (métodos).

Orientação a Objetos

Principais conceitos da OO:

Classes: Moldes para criar objetos.

Objetos: Instâncias de classes.

Encapsulamento: Agrupar dados e métodos relacionados dentro de um objeto.

Herança: Uma classe pode herdar propriedades e métodos de outra classe.

Polimorfismo: Objetos de diferentes classes podem responder ao mesmo método de maneiras diferentes.

Criando Molde para Objetos

- Para criar a classe usamos a palavra-chave **class**.
- Para criar um objeto a partir da classe usamos a palavra chave **new.**
- Quando você usa new, três coisas importantes acontecem:

  Um novo objeto vazio é criado.

  O método **constructor()** da classe é chamado, e o constructor então preenche o objeto com as propriedades iniciais.

  O objeto recém-criado é retornado e atribuído à variável.

Criando Molde para Objetos

O método **constructor()** é usado para inicializar as propriedades do objeto.

Se você não definir um constructor para sua classe, o JavaScript cria um construtor padrão e vazio para você.

Sua principal função é inicializar as propriedades do novo objeto.

O constructor recebe os valores iniciais como parâmetros e os atribui às propriedades do objeto usando a palavra-chave **this.**

Criando Molde para Objetos

Exemplo:

```
class Animal {
constructor(nome) {
this.nome = nome;
}

fazerSom() {
console.log("Som genérico de animal");
}
}
```

Criando Molde para Objetos

Exemplo:

```
let meuAnimal = new Animal("Rex");
console.log(meuAnimal.nome);
meuAnimal.fazerSom();
```

Encapsulamento

Encapsulamento é a prática de agrupar dados (propriedades) e métodos (funções) que agem sobre esses dados em uma única unidade, geralmente um objeto ou classe.

Ele também se refere à ideia de esconder a complexidade interna e restringir o acesso direto a certas propriedades.

Embora JavaScript não tenha propriedades privadas como outras linguagens, o encapsulamento é alcançado por convenção ou com recursos mais recentes.

Encapsulamento

Exemplo:

```
class ContaBancaria {
  constructor(saldoInicial) {
        this._saldo = saldoInicial; // Convenção: sublinhado para propriedades "privadas"
  }
  depositar(valor) {
    if (valor > 0) {
      this._saldo += valor;
      console.log(`Depósito de R$${valor} realizado. `);
    }
 }
  consultarSaldo() {
    return this._saldo;
  }
}
```

Encapsulamento

Exemplo:

```
const minhaConta = new ContaBancaria(100);
minhaConta.depositar(50);
console.log(`Saldo atual: R$${minhaConta.consultarSaldo()} `);
```

Herança

Criação de um "Filho" da Classe

Herança permite que uma classe "filha" (subclasse) herde propriedades e métodos de uma classe "pai" (superclasse).

Isso promove a reutilização de código e cria uma hierarquia lógica entre as classes.

Usamos a palavra-chave **extends** para estabelecer a herança e **super()** para chamar o construtor da classe pai.

Herança

Exemplo:

```
// Classe Pai (Superclasse)
class Veiculo {
  constructor(marca, modelo) {
    this.marca = marca;
    this.modelo = modelo;
  }

  apresentar() {
    console.log(`Este é um ${this.marca} ${this.modelo}.`);
  }
}
```

Herança

Exemplo:

```
// Classe Filha (Subclasse)
class Carro extends Veiculo {
  constructor(marca, modelo, ano) {
    super(marca, modelo); // Chama o construtor do Veiculo
    this.ano = ano;
  }

  acelerar() {
    console.log("O carro está acelerando!");
  }
}
```

Herança

Exemplo:

```
const meuCarro = new Carro("Honda", "Civic", 2022);
meuCarro.apresentar(); // Método herdado da classe pai
meuCarro.acelerar(); // Método da classe filha
```

Polimorfismo

Polimorfismo significa "muitas formas". Na programação, refere-se à capacidade de um mesmo método se comportar de maneira diferente, dependendo do objeto que o chama.

Em JavaScript, isso é facilmente alcançado ao sobrescrever um método da classe pai na classe filha.

Polimorfismo

Exemplo:

```
// Classe Pai
class Animal {
  fazerSom() {
    console.log("Som genérico de animal.");
  }
}

// Classe Filha 1
class Cachorro extends Animal {
  // Sobrescreve o método da classe pai
  fazerSom() {
    console.log("Au au!");
  }
}
```

Polimorfismo

Exemplo:

```
// Classe Filha 2
class Gato extends Animal {
  // Sobrescreve o método da classe pai
  fazerSom() {
    console.log("Miau!");
  }
}
```

Polimorfismo

Exemplo:

```
const meuCachorro = new Cachorro();
const meuGato = new Gato();

// O mesmo método, comportamentos diferentes
meuCachorro.fazerSom(); // Saída: "Au au!"
meuGato.fazerSom();     // Saída: "Miau!"
```

Material de Apoio

- MDN Web Docs (referência oficial).

https://developer.mozilla.org/en-US/docs/Web/JavaScript

typeof e instanceof

typeof informa tipos primitivos;

instanceof verifica protótipo/constructor.

Exemplos:

```
console.log(typeof []); // object
console.log([] instanceof Array); // true
```

Convertendo Tipos

Exemplo:

```
let idade = prompt("Digite sua idade:");
idade = Number(idade);
console.log(typeof idade); // number
```