Programação WEB

React JS
Fundamentos de JavaScript

Laércio Silva
lndsilva@hotmail.com

Fundamentos de JavaScript

Operador Ternário

- O operador condicional (ternário) é o único operador JavaScript que possui três operandos. Este operador é frequentemente usado como um atalho para a instrução if.

- Sintaxe:

condition ? expr1 : expr2

Fundamentos de JavaScript

Exemplo de código utilizando o operador condicional if/else

```
1   let salario = 1000;
2   let bonus = 0.0;
3
4   if (salario > 1000) {
5     bonus = salario * 0.10;
6   } else {
7     bonus = salario * 0.15;
8   }
9
10  console.log(bonus)
```

Fundamentos de JavaScript

Exemplo de código utilizando o operador ternário

```
let salario = 1000;
let bonus = 0.0;

salario > 1000 ? bonus = salario * 0.10 : bonus = salario * 0.15

console.log(bonus)
```

Fundamentos de JavaScript

Operador Spread

- Usado com bastante frequência no React devido à imutabilidade, ou seja, quando criamos um array ou objeto ele não pode ser alterado, sendo assim, precisamos criar um objeto ou array novo a partir do já existente.

- Sintaxe:

  [...]

Fundamentos de JavaScript

Exemplo de código utilizando o operador spread

```
const arr = [0, 1, 2]
const novoArr = [...arr, 3]
console.log(novoArr)
```

Fundamentos de JavaScript

Template String

- Utilizado para fazer a interpolação de strings com expressões de JavaScript, sem necessidade de concatenar as expressões.
- Para escrever um template string é utilizado duas vezes o acento grave (` `) ao invés da comumente utilizada aspas duplas (“ ”).
- Para fazer a interpolação num template string será utilizado a seguinte sintaxe:

  ${ ... }

Fundamentos de JavaScript

Exemplo de concatenação de strings (sem template string)

```
let nome = "João";
let sobrenome = "José";
let texto = "Bem-vindo " + nome + " " + sobrenome + "!";
console.log(texto)
```

Fundamentos de JavaScript

Exemplo de interpolação de strings (com template string)

```
let nome = "João";
let sobrenome = "José";
let texto = `Bem-vindo ${nome} ${sobrenome}!`;
console.log(texto)
```

Fundamentos de JavaScript

Arrow Functions

- A arrow function (=>) é uma sintaxe alternativa, mais compacta quando comparada a uma expressão de função tradicional do JavaScript.

Fundamentos de JavaScript

Exemplo de função com a sintaxe tradicional

```
function total (a, b) {
    return a + b
}
console.log(total(5, 5))
```

Fundamentos de JavaScript

Exemplo de função com a sintaxe arrow function

```
const total = (a, b) => {
    return a + b;
};
console.log(total(5, 5));
```

Fundamentos de JavaScript

É importante ressaltar que essa expressão possui variações da sintaxe, por exemplo, usar o return ou retornar de uma forma implícita.

Variação da sintaxe

```
const total = (a, b) => a + b;
console.log(total(5, 5));
```

Pode-se notar que ao retirarmos as chaves não é mais necessário a utilização do return para encerrar a função.

Fundamentos de JavaScript

Funções anônimas

- É uma função sem nome. Elas são frequentemente utilizadas como argumentos para outras funções (como em callbacks) ou para criar funções rapidamente sem a necessidade de atribuí-las a uma variável.
- Em vez de usar um nome para identificar a função, elas são definidas inline usando a palavra-chave function seguida pelos parênteses () e chaves {} para o corpo da função.

Fundamentos de JavaScript

Funções anônimas

- Exemplo

```
//Passando como callback.

setTimeout(function() {
    console.log("Função anônima executada após 1 segundo");
  }, 1000);
```

Fundamentos de JavaScript

Funções anônimas

- Exemplo

```
//Atribuindo a uma variável.

const minhaFuncao = function(parametro) {
  return parametro * 2;
};

console.log(minhaFuncao(5));
```

Fundamentos de JavaScript

Funções anônimas

- **Arrow functions:** são uma forma concisa de criar funções anônimas, especialmente úteis para funções de uma única linha.

```
//com arrow functions

const dobrar = (numero) => numero * 2;
console.log(dobrar(7));
```

Fundamentos de JavaScript

Objetos

- Em JavaScript, um objeto é uma coleção de propriedades, onde cada propriedade possui um nome (chave) e um valor.
- Esses objetos são como "super variáveis" que podem armazenar múltiplos valores relacionados, facilitando a organização e manipulação de dados.

Fundamentos de JavaScript

Como declarar objetos:

- Existem duas formas principais de declarar objetos em JavaScript:
- **Objeto Literal:** É a forma mais comum e direta de criar um objeto. Utiliza chaves {} para envolver as propriedades, separadas por vírgulas.

```
const pessoa = {
    nome: "João",
    idade: 30,
    cidade: "São Paulo"
};
```

Fundamentos de JavaScript

Acessando propriedades:

Para acessar os valores armazenados em um objeto, utiliza-se a notação de ponto ou a notação de colchetes.

```
// Notação de ponto
console.log(pessoa.nome); // Saída: João

// Notação de colchetes
console.log(pessoa["idade"]); // Saída: 30
```

Fundamentos de JavaScript

Arrays

É um objeto usado para armazenar múltiplos valores em uma única variável, acessados por índices.

Arrays são úteis para organizar e manipular coleções de dados de forma eficiente.

Fundamentos de JavaScript

Como criar um array:

Notação literal: Utilizando colchetes [] e separando os elementos por vírgula

```
let frutas = ["maçã", "banana", "laranja"];
```

Fundamentos de JavaScript

Construtor Array: Usando o construtor new Array()

```
let numeros = new Array(1, 2, 3, 4, 5);
```

É importante notar que, ao passar apenas um número para o construtor new Array(), ele cria um array com o tamanho especificado.

Fundamentos de JavaScript

Acessando elementos:

Cada elemento em um array é acessado por seu índice, que começa em 0.

```
let primeiroItem = frutas[0]; // Acessa "maçã"

let segundoItem = numeros[1]; // Acessa 2
```

Fundamentos de JavaScript

Acessando elementos de um array usando estruturas de repetição

Podemos os loops for, while ou forEach.

O loop for é a forma mais comum, permitindo percorrer o array com base em um índice.

O loop while é útil quando o número de iterações não é conhecido antecipadamente.

O método forEach oferece uma sintaxe mais concisa para iterar sobre os elementos do array.

Fundamentos de JavaScript

1. Loop for:

```
let frutas = ["maçã", "banana", "laranja"];

for (let i = 0; i < frutas.length; i++) {
  console.log(frutas[i]);
}
```

Fundamentos de JavaScript

2. Loop while:

```
let numeros = [10, 20, 30, 40, 50];
let i = 0;

while (i < numeros.length) {
  console.log(numeros[i]);
  i++;
}
```

Fundamentos de JavaScript

3. Método forEach:

```
let cores = ["vermelho", "verde", "azul"];

cores.forEach(function(cor) {
  console.log(cor);
});
```

forEach itera sobre cada elemento do array.

A função dentro do forEach recebe cada elemento como argumento (cor neste caso) e executa o código especificado.

Fundamentos de JavaScript

Métodos de array

- Utilizando o React, será muito comum utilizar métodos de array, principalmente o map, filter, reduce e find.
- Eles são métodos puros, que recebem um valor e retornam outro da forma desejada.

Fundamentos de JavaScript

Map

- É utilizada para percorrer cada item de um array, recebe uma função como callback ou apenas utiliza uma função anônima, que recebe o item como parâmetro para ser utilizado.
- O map retorna um novo array com as modificações da função de callback aplicadas.

Fundamentos de JavaScript

Map

```
const map1 = array1.map((x) => x * 2);

console.log(map1);
```

Fundamentos de JavaScript

Filter

- O método filter() cria um novo array com todos os elementos que passaram no teste implementado pela função fornecida.
- filter() chama a função callback fornecida, uma vez para cada elemento do array, e constrói um novo array com todos os valores para os quais o callback retornou o valor true ou um valor que seja convertido para true.
- O callback é chamado apenas para índices do array que possuem valores atribuídos;
- Ele não é invocado para índices que foram excluídos ou para aqueles que não tiveram valor atribuído.
- Elementos do array que não passaram no teste do callback são simplesmente ignorados, e não são incluídos no novo array.

Fundamentos de JavaScript

Filter

```
const words = ["spray", "elite", "exuberant", "destruction", "present"];

const result = words.filter((word) => word.length > 6);

console.log(result);
```

Fundamentos de JavaScript

Reduce

- Passa por cada item do array fazendo uma expressão escolhida, e no final vai devolver um único valor.
- Esse método extrai informações únicas de uma lista e pode ser utilizado para extrair o maior ou menor valor, a média, soma, e assim por diante.
- Esse método pode ser utilizado em diversos cenários. Por exemplo, imagine que você trabalhe com uma empresa de vendas e tenha um banco de dados ou planilha com o registro de vendas dos produtos. Com esse método, você consegue descobrir o maior valor de vendas, o menor valor, o faturamento total, entre outras informações relevantes.

Fundamentos de JavaScript

Reduce – Maior valor

```
const vendas = [650, 550, 1020, 1060, 200, 150, 495, 875];

const maiorValor = vendas.reduce((maiorValor, elementoAtual) => {
  return maiorValor > elementoAtual ? maiorValor : elementoAtual;
}, 0);

console.log(maiorValor)
```

Fundamentos de JavaScript

Reduce – Maior valor

Nossa arrow function recebe dois parâmetros: maiorValor, que inicia como sendo 0, e elementoAtual, o valor do array que está sendo percorrido no momento.

A partir disso, comparamos se o maiorValor é maior do que o elementoAtual. Se isso for verdadeiro, ela retornará o maiorValor, caso contrário, retornará o elementoAtual.

Então, após a primeira iteração da função, o valor da variável maiorValor passará a ser 650, porque 650 é maior do que 0. Na segunda iteração, ela continuará sendo 650, isso porque 550 é menor do que 650. Já na terceira iteração, ela será atualizada para 1020.

Fundamentos de JavaScript

Reduce – Soma valor

```
const vendas = [650, 550, 1020, 1060, 200, 150, 495, 875];

const faturamentoDiario = vendas.reduce((valorAcumulado, elementoAtual) => {
  return valorAcumulado + elementoAtual;
}, 0);

console.log(faturamentoDiario);
```

Fundamentos de JavaScript

Reduce – Soma valor

Repare que a lógica é bem semelhante à do maior valor. Definimos dois parâmetros para a nossa função, que serão o valorAcumulado e o elementoAtual, e o valor inicial do método reduce como 0.

Na primeira iteração, o valorAcumulado será 0 e será somado a ele o elementoAtual, que é 650. Assim, o valorAcumulado passa a ser 650 na segunda iteração, onde é somado 550 do elementoAtual. Fazendo isso para cada um dos valores no array.

Executando esse código, teremos o somatório final, ou seja, o faturamento diário dessa empresa.

Fundamentos de JavaScript

Reduce – Menor valor

```
const vendas = [650, 550, 1020, 1060, 200, 150, 495, 875];

const menorValor = vendas.reduce((menorValor, elementoAtual) => {
  return menorValor < elementoAtual ? menorValor : elementoAtual;
});

console.log(menorValor);
```

Fundamentos de JavaScript

Reduce – Menor valor

A lógica é praticamente a mesma do maior valor, mas ao invés de comparar se o maiorValor é maior do que o elementoAtual, verificamos se o menorValor é menor do que o elementoAtual.

Além disso, repare que para esse caso nós não definimos o valor inicial como 0. Caso fizéssemos isso, o resultado retornado seria 0, pois não existe um valor dentro do array que seja menor do que 0.

O valor inicial não precisa ser definido para que o método reduce funcione; ele é um argumento opcional. Quando não definimos um valor inicial, o método considera o primeiro valor do array como sendo o valor inicial.

Fundamentos de JavaScript

Reduce – Média de valores

```
const vendas = [650, 550, 1020, 1060, 200, 150, 495, 875];

const mediaVendedores =
  vendas.reduce((valorAcumulado, elementoAtual) => {
    return valorAcumulado + elementoAtual;
  }, 0) / vendas.length;

console.log(mediaVendedores);
```

Fundamentos de JavaScript

Reduce – Média de valores

Pode-se calcular o valor da média de vendas por vendedor ao longo desse dia. A média é obtida calculando a soma total das vendas dividia pelo número de funcionários.

Como cada valor dentro do array representa as vendas de um funcionário, temos que o número de funcionários da empresa será igual à propriedade length do array.

Essa propriedade retorna o número de elementos em um array.

Então, para calcularmos a média de valores, basta aplicarmos o método reduce para obter o faturamento diário, como já fizemos anteriormente, e dividir o valor retornado (soma das vendas) pela propriedade length.

Vamos
para
os exercícios