7. Fundamentos de JavaScript

JavaScript é uma das linguagens de programação mais utilizadas para o desenvolvimento de aplicações web interativas e dinâmicas. Criada por Brendan Eich em 1995, a linguagem se tornou essencial para a criação de comportamentos interativos nas páginas da web, tornando-as mais responsivas e proporcionando uma experiência mais rica aos usuários.

Conforme documentado pela W3C (World Wide Web Consortium), uma das principais organizações internacionais que desenvolve padrões para a web, JavaScript é uma linguagem de script client-side amplamente adotada pelos navegadores modernos. Ela permite a manipulação do DOM (Document Object Model) e a interação com os elementos da página, possibilitando alterações de conteúdo e estilos de forma dinâmica, sem a necessidade de recarregar a página.

7.1. Introdução ao JavaScript: variáveis, tipos de dados, operadores

Uma das características fundamentais do JavaScript é a capacidade de trabalhar com variáveis. De acordo com a especificação ECMAScript, uma variável é uma unidade de armazenamento de informações que pode conter diferentes tipos de dados. Elas são declaradas utilizando as palavras-chave var, let ou const, sendo que let e const foram introduzidas a partir da versão ECMAScript 6.

Os tipos de dados em JavaScript são divididos em primitivos e objetos. Os primitivos incluem números, strings, booleanos, null e undefined, enquanto os objetos são estruturas de dados mais complexas, como arrays e objetos literais.

Além das variáveis e tipos de dados, o JavaScript suporta uma variedade de operadores para realizar operações matemáticas, comparações e atribuições.

Exemplos de operadores aritméticos são +, -, * e /, enquanto os operadores de comparação incluem ==, !=, >, <, entre outros.

Declaração de Variáveis:

Em JavaScript, as variáveis são usadas para armazenar e manipular dados. Elas podem ser declaradas utilizando três palavras-chave: var, let ou const. A diferença entre elas está na sua abrangência (escopo) e na possibilidade de reatribuição do valor.
Exemplo de declaração de variáveis:

```javascript
// Utilizando var (escopo de função)
var idade = 25;
var nome = 'Maria';

// Utilizando let (escopo de bloco)
let quantidade = 10;
let produto = 'Camiseta';

// Utilizando const (valor constante)
const pi = 3.14;
const url = 'https://www.example.com';
```

Tipos de Dados:

JavaScript é uma linguagem dinamicamente tipada, o que significa que não é necessário definir explicitamente o tipo de dado ao declarar uma variável. Os tipos de dados são determinados automaticamente com base no valor atribuído.

Principais tipos de dados em JavaScript:

Números: podem ser inteiros ou decimais.

Strings: sequências de caracteres, devem estar entre aspas simples ou duplas.

Booleanos: representam valores verdadeiros (true) ou falsos (false).

Null e undefined: representam valores nulos e indefinidos, respectivamente.

Objetos: estruturas de dados mais complexas, como arrays e objetos literais.

Exemplo de uso de tipos de dados:

```javascript
// Números
let idade = 30;
let altura = 1.75;

// Strings
let nome = 'João';
let sobrenome = "Silva";

// Booleanos
let ehMaiorDeIdade = true;
let temContaBancaria = false;

// null e undefined
let valorNulo = null;
let valorIndefinido;

// Objetos
let listaDeCompras = ['Maçã', 'Banana', 'Pão'];
let pessoa = { nome: 'Maria', idade: 25 };
```

Operadores:

Em JavaScript, os operadores são usados para realizar operações matemáticas, comparações e atribuições.

Principais operadores aritméticos:

+ (adição)

- (subtração)

***** (multiplicação)

/ (divisão)

% (resto da divisão)

Principais operadores de comparação:

== (igual)
!= (diferente)
> (maior que)
< (menor que)
>= (maior ou igual)
<= (menor ou igual)

Exemplo de uso de operadores:

```javascript
let numero1 = 10;
let numero2 = 5;

let soma = numero1 + numero2; // 10 + 5 = 15
let subtracao = numero1 - numero2; // 10 - 5 = 5
let multiplicacao = numero1 * numero2; // 10 * 5 = 50
let divisao = numero1 / numero2; // 10 / 5 = 2
let restoDivisao = numero1 % numero2; // 10 % 5 = 0

let ehMaior = numero1 > numero2; // true
let ehMenor = numero1 < numero2; // false
let igual = numero1 == numero2; // false
```

7.2. Estruturas de controle: condicionais e loops

As estruturas de controle são fundamentais na programação, permitindo que os desenvolvedores controlem o fluxo de execução do código com base em condições ou para repetir determinadas instruções. Em JavaScript, duas estruturas de controle importantes são as condicionais e os loops.

Condicionais:

As estruturas condicionais permitem que o programa tome decisões com base em condições especificadas. O principal tipo de estrutura condicional em JavaScript é o if, que executa um bloco de código se a condição especificada for verdadeira.

Exemplo de uso do if:

```JavaScript
let idade = 18;

if (idade >= 18) {
  console.log('Você é maior de idade.');
} else {
  console.log('Você é menor de idade.');
}
```

Além do if, existem outras formas de estruturas condicionais em JavaScript, como o else if, que permite especificar múltiplas condições encadeadas.

Loops:

Os loops permitem que uma determinada seção de código seja executada repetidamente enquanto uma condição for verdadeira. Em JavaScript, existem dois tipos principais de loops: o for e o while.

Exemplo de uso do for:

```JavaScript
for (let i = 1; i <= 5; i++) {
  console.log('Número: ' + i);
}
```

Nesse exemplo, o loop for é utilizado para imprimir os números de 1 a 5 na saída do console.

Exemplo de uso do while:

```javascript
let contador = 1;

while (contador <= 5) {
  console.log('Contador: ' + contador);
  contador++;
}
```

O loop while também imprime os números de 1 a 5, mas utiliza uma abordagem um pouco diferente em relação ao for.

Break e Continue:

Em loops, é possível utilizar as palavras-chave **break** e **continue** para controlar o fluxo de execução. O **break** interrompe a execução do loop imediatamente, enquanto o **continue** pula para a próxima iteração do loop.

Exemplo de uso do break:

```javascript
for (let i = 1; i <= 10; i++) {
  if (i == 5) {
    break;
  }
  console.log('Número: ' + i);
}
```

Nesse exemplo, o loop for será interrompido quando o valor da variável i for igual a 5.

```javascript
for (let i = 1; i <= 10; i++) {
 if (i == 5) {
  continue;
 }
 console.log(i);
}
```

Neste exemplo, usamos um loop for para iterar de 1 a 10. Quando o valor de i é igual a 5, a instrução **continue** é acionada, interrompendo a iteração atual e passando para a próxima iteração. Isso resultará em imprimir todos os números de 1 a 10, exceto o número 5.

7.3. Funções em JavaScript: criação e uso

As funções são blocos de código reutilizáveis que executam uma tarefa específica. Elas são essenciais na programação, permitindo que um conjunto de instruções seja agrupado sob um único nome e chamado repetidamente conforme necessário. Em JavaScript, as funções são objetos de primeira classe, o que significa que podem ser atribuídas a variáveis, passadas como argumentos para outras funções e retornadas como valores de outras funções.

Criação de Funções:

Em JavaScript, as funções podem ser criadas de várias maneiras. A forma mais comum é usando a palavra-chave **function,** seguida pelo nome da função e os parâmetros entre parênteses. O corpo da função é delimitado por chaves **{}** e contém as instruções a serem executadas quando a função for chamada.

Exemplo de criação de função:

```JavaScript
function saudacao(nome) {
  return 'Olá, ' + nome + '!';
}
```

Neste exemplo, criamos uma função chamada saudacao que recebe um parâmetro nome e retorna uma saudação personalizada.

Chamada de Funções:

Para chamar uma função em JavaScript, basta digitar o nome da função seguido pelos parênteses contendo os argumentos, se houver. Os argumentos são os valores que a função receberá quando for chamada.

Exemplo de chamada de função:

```JavaScript
let resultado = saudacao('João');
console.log(resultado); // Saída: "Olá, João!"
```

No exemplo acima, chamamos a função saudacao com o argumento 'João' e atribuímos o resultado retornado pela função à variável resultado.

Funções Anônimas e Arrow Functions:

Além da forma tradicional de criação de funções, JavaScript também suporta funções anônimas e arrow functions, que são formas mais compactas e expressivas de declarar funções.

Exemplo de função anônima:

```JavaScript
let saudacao = function(nome) {
  return 'Olá, ' + nome + '!';
};
```

Exemplo de arrow function:

```JavaScript
let saudacao = nome => 'Olá, ' + nome + '!';
```

As variáveis declaradas dentro de uma função têm escopo local, o que significa que elas só podem ser acessadas dentro da própria função. Variáveis declaradas fora de uma função têm escopo global, podendo ser acessadas em qualquer parte do código.