Desenvolvimento Web

Javascript

Índice

Introdução ao JavaScript

JavaScript é uma linguagem de programação de alto nível, interpretada e orientada a objetos. Ela é amplamente utilizada para tornar páginas web dinâmicas e interativas, permitindo manipulação de elementos HTML, processamento de eventos do usuário e comunicação com servidores.

Incorporando JavaScript no HTML

JavaScript pode ser incorporado diretamente em um documento HTML usando a tag <script>. Isso pode ser feito dentro da seção <head> ou <body>. Alternativamente, também é possível vincular arquivos JavaScript externos usando a tag <script> com o atributo src.

Comentários

Comentários em JavaScript são precedidos por // para comentários de uma linha, ou envolvidos entre /* e */ para comentários de várias linhas. Eles são úteis para explicar o código e facilitar a manutenção.

Variáveis

Em JavaScript, variáveis são espaços reservados na memória para armazenar valores ou referências a valores. Elas são fundamentais para o desenvolvimento, permitindo que os desenvolvedores armazenem e manipulem dados de maneira dinâmica. No entanto, até o ECMAScript 6 (ES6), JavaScript não tinha suporte nativo para variáveis no sentido tradicional. O conceito de variáveis em JavaScript foi ampliado com a introdução das variáveis com a palavra-chave let e a adição de constantes usando const no ES6. Exemplo:

```
let numero = 20;

const nome = “Dissoloquele”;
```

Estruturas de controle de fluxo

As estruturas de controle de fluxo em JavaScript são ferramentas essenciais para direcionar o comportamento do programa com base em condições específicas. Aqui estão as principais estruturas de controle de fluxo em JavaScript:

Estrutura Condicional if:

A estrutura if permite que você execute um bloco de código se uma condição especificada for verdadeira. Exemplo:

Estruturas de controle de fluxo

Estrutura Condicional else if: else if é usado para verificar várias condições encadeadas.

```
let nota = 75;

if (nota >= 90) {
  console.log("A");
} else if (nota >= 80) {
  console.log("B");
} else if (nota >= 70) {
  console.log("C");
} else {
  console.log("D");
}
```

```
let idade = 18;

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

Estruturas de controle de fluxo

Estrutura de Loop for: for é usado para iterar sobre uma sequência de valores ou realizar uma ação um número específico de vezes. Exemplo:

```
for (let i = 0; i < 5; i++) {

  console.log(i);

}
```

Estrutura de Loop while executa um bloco de código enquanto uma condição especificada for verdadeira. Exemplo:

Estruturas de controle de fluxo

```
let contador = 0;

while (contador < 5) {
  console.log(contador);
  contador++;
}
```

Estrutura de Loop do...while:

do...while é semelhante ao while, mas garante que o bloco de código seja executado pelo menos uma vez, mesmo se a condição for falsa inicialmente. Exemplo:

Estruturas de controle de fluxo

```
let contador = 0;


do {
  console.log(contador);
  contador++;
} while (contador < 5);
```

Eventos

Em JavaScript, eventos são ações ou ocorrências que acontecem durante a execução de uma página web, como cliques do mouse, pressionamentos de teclas, carregamento da página, entre outros. Esses eventos são fundamentais para criar interatividade dinâmica em uma aplicação web. A manipulação de eventos é realizada por meio de event listeners, que são funções associadas a elementos HTML específicos e que são acionadas quando o evento ocorre. Por exemplo, para lidar com um clique do mouse, você pode utilizar o evento "click" e associar uma função ao elemento desejado usando addEventListener.

Eventos

Para utilizar eventos em JavaScript, primeiro, identifique o elemento HTML alvo e, em seguida, adicione um event listener para esse elemento. O event listener pode ser configurado para chamar uma função específica quando o evento ocorre. Por exemplo, para capturar um clique do mouse em um botão com o ID "meuBotao", você pode usar:

```
const botao = document.getElementById('meuBotao');

botao.addEventListener('click', function() {
  // Código a ser executado quando o botão for clicado
  console.log('Botão clicado!');
});
```

Eventos

- **click**: Ocorre quando um elemento é clicado com o mouse.
- **keydown**: Disparado quando uma tecla é pressionada no teclado.
- **keyup**: Acionado quando uma tecla do teclado é liberada.
- **mouseover**: Ativado quando o ponteiro do mouse entra em um elemento.
- **mouseout**: Disparado quando o ponteiro do mouse deixa um elemento.
- **change**: Ocorre quando o valor de um elemento de formulário (como um <input>, <select>, ou <textarea>) é alterado.
- **submit**: Acionado quando um formulário é enviado.
- **focus**: Disparado quando um elemento recebe o foco.
- **blur**: Ativado quando um elemento perde o foco.
- **load**: Ocorre quando um recurso e seus recursos dependentes terminaram de

Eventos

- **unload**: Disparado quando a página está prestes a ser descarregada (quando o usuário está saindo da página).
- **scroll**: Acionado quando a posição de rolagem de um elemento é alterada.
- **resize**: Ocorre quando a janela do navegador é redimensionada.
- **input**: Disparado a cada vez que o valor de um elemento de entrada muda. Funciona bem com elementos de formulário como <input>, <select>, e <textarea>.
- **contextmenu**: Acionado quando o botão direito do mouse é clicado sobre um elemento, abrindo o menu de contexto.

Esses eventos são fundamentais para a manipulação e interação do usuário em páginas web, oferecendo aos desenvolvedores a capacidade de criar interfaces dinâmicas e responsivas.

DOM

O Document Object Model (DOM) em JavaScript é uma representação em árvore da estrutura HTML ou XML de uma página web, permitindo a interação dinâmica e manipulação dos elementos presentes nela. O DOM converte a estrutura do documento em uma hierarquia de objetos acessíveis através do JavaScript, onde cada elemento, atributo e texto é representado por um nó no DOM. Essa representação em árvore possibilita que os desenvolvedores alterem o conteúdo, estilo e comportamento da página de forma dinâmica, respondendo a eventos do usuário e atualizando a interface em tempo real.

Para utilizar o DOM em JavaScript, os desenvolvedores podem selecionar elementos, modificar atributos, adicionar ou remover elementos, e responder a eventos. Métodos como getElementById, querySelector, appendChild e addEventListener são comumente empregados. O DOM é essencial para criar aplicações web interativas, pois permite a manipulação direta da estrutura da página, proporcionando uma experiência de usuário dinâmica e responsiva. É fundamental compreender o DOM para desenvolver de forma eficaz e criar interfaces web dinâmicas e interativas.

DOM - Manipulação do HTML

document.getElementById(id) : Retorna uma referência ao elemento com o ID especificado. Ex:

const meuElemento = document.getElementById('meuId');

document.getElementsByClassName(className): Retorna uma coleção de elementos com a classe CSS especificada.

const meusElementos = document.getElementsByClassName('minhaClasse');

document.getElementsByTagName(tagName): Retorna uma coleção de elementos com a tag HTML especificada.

const minhasTags = document.getElementsByTagName('div');

document.querySelector(selector): Retorna o primeiro elemento que corresponde ao seletor CSS especificado.

const meuElemento = document.querySelector('#meuId');

DOM - Manipulação do HTML

document.querySelectorAll(selector): Retorna todos os elementos que correspondem ao seletor CSS especificado.Ex:

const meusElementos = document.querySelectorAll('.minhaClasse');

element.innerHTML: Obtém ou define o conteúdo HTML de um elemento. Ex:

meuElemento.innerHTML = ' Novo conteúdo ';

element.textContent: Obtém ou define o texto de um elemento.

meuElemento.textContent = 'Novo texto';

DOM - Manipulação do CSS

element.style.property: Acessa ou define diretamente as propriedades de estilo de um elemento.

meuElemento.style.color = 'blue';

meuElemento.style.fontSize = '16px';

element.classList: Fornece acesso às classes de um elemento para adicionar, remover ou alternar classes. Ex:

meuElemento.classList.add('destaque'); meuElemento.classList.remove('inativo');

element.setAttribute(name, value): Define o valor de um atributo em um elemento.

meuElemento.setAttribute('class', 'novoEstilo');

element.removeAttribute(name): Remove um atributo de um elemento. Ex.:

meuElemento.removeAttribute('class');

Funções

Em JavaScript, funções desempenham um papel fundamental ao permitir a modularidade e reutilização de código. Uma função é um bloco de código designado para executar uma tarefa específica, podendo aceitar argumentos e retornar valores. Elas são essenciais para organizar o código de maneira eficiente, facilitando a manutenção e compreensão do programa. Além disso, as funções podem ser definidas e chamadas em diferentes partes do código, promovendo a abstração e encapsulamento de lógica, o que contribui para a construção de aplicações mais estruturadas e flexíveis.

Ao criar funções em JavaScript, os desenvolvedores podem encapsular operações específicas, melhorando a legibilidade e promovendo a reutilização de código. Essa abstração é crucial para o desenvolvimento web, onde a interatividade e dinamicidade são fundamentais. Ao compreender e aplicar o conceito de funções, os programadores podem criar aplicações mais eficientes e manuteníveis, resultando em uma experiência de desenvolvimento web mais coesa e robusta.

Funções

Criando uma Função: Para criar uma função em JavaScript, use a palavra-chave function, seguida pelo nome da função e, se necessário, parâmetros entre parênteses. Ex:

```
function saudacao(nome) {

  console.log('Olá, ' + nome + '!');

}
```

Funções

Chamando uma Função: Chame uma função fornecendo seu nome seguido por parênteses e, se necessário, os valores dos parâmetros. Ex:

saudacao('João'); // Resultado: Olá, João!

Parâmetros e Retorno: Funções podem receber parâmetros (informações) e retornar um resultado usando return.

Funções

```
function soma(a, b) {
  return a + b;
}

const resultado = soma(3, 5);

console.log(resultado); // Resultado: 8
```

Funções Anônimas e Expressões de Função: Você pode criar funções sem nome, chamadas de funções anônimas, para uso em expressões.

Funções

Ex:

```
const multiplicacao = function(x, y) {
  return x * y;
};
const resultadoMultiplicacao = multiplicacao(4, 3);
console.log(resultadoMultiplicacao); // Resultado: 12
```

Conclusão

Neste guia introdutório de JavaScript, exploramos os fundamentos essenciais, desde variáveis e estruturas de controle até a manipulação do DOM para criar páginas web dinâmicas. Destacamos a importância das funções para modularização e reutilização de código. Esses conhecimentos formam a base para o desenvolvimento web interativo. À medida que você avança, explore conceitos avançados e frameworks modernos para aprimorar suas habilidades. O JavaScript oferece um mundo de possibilidades, e a prática contínua é fundamental para se tornar um desenvolvedor web proficient.