

MÓDULO VII

Lógica de Programação utilizando a Linguagem JavaScript

MÓDULO VII

Revisão de JavaScript

Em primeiro lugar, compreenderemos o passo mais simples, a **sintaxe básica do JavaScript** que é relativamente simples e fácil de entender. Para iniciar, você precisa ter um arquivo **HTML** onde irá incluir seu código **JavaScript**. Você pode fazer isso usando a tag dentro do seu arquivo **HTML**.
Dentro dessa tag, você pode escrever seu código **JavaScript**.

Um dos elementos mais básicos deste programa é a declaração de variáveis.

Você pode declarar uma variável usando a palavra-chave “**var**” seguida pelo nome da variável.

Por exemplo, você pode declarar uma variável chamada “**idade**” da seguinte forma:

MÓDULO VII

var idade;

Após declarar uma **variável**, você pode atribuir um valor a ela usando o operador de atribuição “=”.

idade = 25;

Você também pode declarar e atribuir um valor a uma variável ao mesmo tempo:

var nome = “João”;

Além disso, o **JavaScript** possui uma série de operadores aritméticos, como **adição** (+), **subtração** (-), **multiplicação** (*) e **divisão** (/), que podem ser usados para realizar cálculos.

MÓDULO VII

Outro conceito importante é a função. Uma **função** é um bloco de código que pode ser chamado e executado várias vezes. Você pode criar uma função usando a palavra-chave “**function**” seguida pelo nome da **função** e seus **parâmetros**. Por exemplo, você pode criar uma função chamada “**saudacao**” que recebe um nome como **parâmetro** e exibe uma **saudação**:

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

Após criar uma função, você pode chamá-la passando os argumentos necessários:

MÓDULO VII

```
saudacao("Maria");
```

Isso exibirá a mensagem “Olá, Maria!” no console.

Esses são apenas alguns dos conceitos básicos do JavaScript. À medida que você avança em seu aprendizado, você descobrirá muitos outros recursos e funcionalidades poderosas que o **JavaScript** tem a oferecer. ***Continue a leitura para compreender os 5 principais conceitos que todo programador deve saber.***

MÓDULO VII

1. Variáveis e tipos de dados

Variáveis e tipos de dados são conceitos fundamentais em JavaScript. Uma variável é um contêiner que contém um valor e pode receber diferentes valores ao longo do programa. **Em JavaScript, existem vários tipos de dados, incluindo números, *strings*, *booleanos*, *arrays* e objetos.**

Compreender como declarar e usar variáveis, bem como trabalhar com diferentes tipos de dados, é essencial para escrever um código **JavaScript** eficaz. Ao dominar esses conceitos, você conseguirá manipular e armazenar dados eficientemente, tornando seu código mais dinâmico e flexível.

MÓDULO VII

2. Funções e Escopo

Funções e escopo são conceitos cruciais em JavaScript que todo desenvolvedor deve entender. Uma função é um bloco de código que executa uma tarefa específica e pode ser reutilizado em todo o programa. Ele permite que você organize seu código e o torne mais modular e reutilizável. Em **JavaScript**, as funções podem ser declaradas usando a palavra-chave ***function*** e podem receber parâmetros e valores de retorno.

MÓDULO VII

Escopo refere-se à visibilidade e acessibilidade de variáveis em um programa.

JavaScript tem dois tipos de escopo: escopo global e escopo local.

As variáveis declaradas fora de qualquer função têm escopo global e podem ser acessadas de qualquer lugar do programa. Por outro lado, variáveis declaradas em uma função possuem escopo local e só podem ser acessadas dentro dessa função.

Entender como as funções e o escopo funcionam juntos é essencial para escrever um código JavaScript limpo e eficiente. Ele permite encapsular a lógica, evitar conflitos de nomenclatura e controlar a visibilidade das variáveis. Ao dominar esses conceitos, você poderá escrever códigos mais organizados e sustentáveis, tornando seus aplicativos JavaScript mais robustos e escaláveis.

MÓDULO VII

3. Matrizes e Objetos

Arrays e **objetos** são estruturas de dados fundamentais em **JavaScript** que permitem armazenar e manipular coleções de valores.

Uma matriz é uma lista ordenada de valores, que pode ser de qualquer tipo de dados.

Você pode acessar elementos individuais de um array usando seu índice, começando em **0**.

Arrays em **JavaScript** são dinâmicos, o que significa que você pode adicionar ou remover elementos conforme necessário. Isso os torna versáteis para armazenar e manipular dados.

MÓDULO VII

Os objetos, por outro lado, são coleções de pares chave-valor. Cada chave é um identificador exclusivo, também conhecido como propriedade, e cada valor pode ser de qualquer tipo de dados. Os objetos em **JavaScript** costumam ser usados para representar entidades reais ou estruturas de dados complexas. ***Você pode acessar as propriedades de um objeto usando notação de ponto ou notação de colchetes.***

Entender como trabalhar com **arrays** e **objetos** é crucial para manipular e organizar dados em **JavaScript**. Eles fornecem ferramentas poderosas para armazenar e acessar informações, permitindo que você crie aplicativos mais complexos e dinâmicos.

Portanto, ao dominar esses conceitos, você poderá lidar com dados de maneira mais eficiente e eficaz em seu código **JavaScript**.

MÓDULO VII

4. Fluxo de controle e loops

Fluxo de controle e loops são conceitos essenciais em JavaScript que permitem controlar o fluxo do código e repetir determinadas ações.

O fluxo de controle refere-se à ordem na qual as instruções são executadas em um programa. O **JavaScript** fornece várias instruções de fluxo de controle, como instruções ***if-else*** e instruções ***switch***, que permitem que você tome decisões com base em determinadas condições.

Por exemplo, **you can use an *if-else* instruction to execute a specific block of code** se uma condição for verdadeira e um bloco de código diferente se a condição for falsa.

MÓDULO VII

Os loops, por outro lado, permitem que você repita um bloco de código várias vezes. JavaScript fornece diferentes tipos de loops, incluindo ***loops for***, ***loops while*** e ***loops do-while***. Por exemplo, você pode usar um **loop for** para iterar em uma matriz e executar uma determinada ação em cada elemento.

Compreendendo o fluxo de controle e os loops, você pode criar programas JavaScript mais dinâmicos e interativos. Você pode controlar a execução do seu código com base em diferentes condições e repetir determinadas ações conforme necessário. Isso permite que você crie mais eficientemente e aplicativos flexíveis que podem lidar com uma variedade de cenários.

MÓDULO VII

5. Manipulação de DOM

A manipulação de **DOM** é um conceito crucial em **JavaScript** que permite interagir e modificar a estrutura **HTML** de uma página da web. O *Document Object Model* (**DOM**) representa a estrutura de um documento **HTML** como uma estrutura semelhante a uma árvore, com cada elemento representado como um nó.

Com a manipulação do **DOM**, você pode acessar e modificar elementos, atributos e conteúdo em uma página da web. Por exemplo, você pode usar **JavaScript** para adicionar ou remover elementos dinamicamente, alterar o texto ou o estilo dos elementos e responder às interações do usuário.

Para manipular o **DOM**, você pode usar vários métodos e propriedades fornecidos pelo objeto **Document**, como ***getElementById***, ***getElementsByClassName*** e ***querySelector***. Esses métodos permitem selecionar elementos específicos em uma página da Web com base em seus **IDs**, classes ou outros atributos.

MÓDULO VII

Após selecionar um elemento, você pode usar propriedades como ***innerHTML*** e **style** para modificar seu conteúdo e estilo. Você também pode usar métodos como ***appendChild*** e ***removeChild*** para adicionar ou remover elementos do **DOM**.

Ao dominar a manipulação do **DOM**, você pode criar páginas da Web dinâmicas e interativas que respondem às ações do usuário sendo atualizadas em tempo real. Isso é essencial para criar aplicativos da Web modernos e aprimorar a experiência do usuário.

MÓDULO VII

Os 4 pilares da Programação Orientada a Objetos

Para entendermos exatamente do que se trata a orientação a objetos, vamos entender quais são os requerimentos de uma linguagem para ser considerada nesse paradigma. Para isso, a linguagem precisa atender a quatro tópicos bastante importantes:

Abstração

A abstração consiste em um dos pontos mais importantes dentro de qualquer linguagem **Orientada a Objetos**. Como estamos lidando com uma representação de um objeto real (o que dá nome ao paradigma), temos que imaginar o que esse objeto irá realizar dentro de nosso sistema. São três pontos que devem ser levados em consideração nessa abstração.

MÓDULO VII

O primeiro ponto é darmos uma **identidade** ao objeto que iremos criar. Essa identidade deve ser única dentro do sistema para que não haja conflito. Na maior parte das linguagens, há o conceito de pacotes (ou **namespaces**).

Nessas linguagens, a identidade do objeto não pode ser repetida dentro do pacote, e não necessariamente no sistema inteiro..

A segunda parte diz respeito a características do objeto. Como sabemos, no mundo real qualquer objeto possui elementos que o definem. Dentro da programação orientada a objetos, essas características são nomeadas **propriedades**. Por exemplo, as propriedades de um objeto “**Cachorro**” poderiam ser “**Tamanho**”, “**Raça**” e “**Idade**”.

Por fim, a terceira parte é definirmos as ações que o objeto irá executar. Essas ações, ou eventos, são chamados **métodos**. Esses métodos podem ser extremamente variáveis, desde “**Acender()**” em um objeto lâmpada até “**Latir()**” em um objeto cachorro.

MÓDULO VII

Encapsulamento

O **encapsulamento** é uma das principais técnicas que define a programação orientada a objetos. Se trata de um dos elementos que adicionam segurança à aplicação em uma programação orientada a objetos pelo fato de esconder as propriedades, criando uma espécie de caixa preta.

A maior parte das linguagens orientadas a objetos implementam o encapsulamento baseado em propriedades privadas, ligadas a métodos especiais chamados **getters** e **setters**, que irão retornar e setar o valor da propriedade, respectivamente. Essa atitude evita o acesso direto a propriedade do objeto, adicionando uma outra camada de segurança à aplicação.

Para fazermos um paralelo com o que vemos no mundo real, temos o encapsulamento em outros elementos. Por exemplo, quando clicamos no botão ligar da televisão, não sabemos o que está acontecendo internamente. Podemos então dizer que os métodos que ligam a televisão estão encapsulados.

MÓDULO VII

Herança

O reuso de código é uma das grandes vantagens da programação orientada a objetos.

Muito disso se dá por uma questão que é conhecida como *herança*.

Essa característica otimiza a produção da aplicação em tempo e linhas de código.

Para entendermos essa característica, vamos imaginar uma família: a criança, por exemplo, está herdando características de seus pais. Os pais, por sua vez, herdam algo dos avós, o que faz com que a criança também o faça, e assim sucessivamente. O objeto abaixo na hierarquia irá herdar características de todos os objetos acima dele, seus “ancestrais”. A herança a partir das características do objeto mais acima é considerada herança direta, enquanto as demais são consideradas heranças indiretas. Por exemplo, na família, a criança herda diretamente do pai e indiretamente do avô e do bisavô.

MÓDULO VII

A questão da herança varia bastante de linguagem para linguagem. Em algumas delas, como C++, há a questão da herança múltipla. Isso, essencialmente, significa que o objeto pode herdar características de vários “ancestrais” ao mesmo tempo diretamente. Em outras palavras, cada objeto pode possuir quantos pais for necessário. Devido a problemas, essa prática não foi difundida em linguagens mais modernas, que utilizam outras artimanhas para criar uma espécie de herança múltipla.

Outras linguagens orientadas a objetos, como C#, trazem um objeto base para todos os demais. A classe *object* fornece características para todos os objetos em C#, sejam criados pelo usuário ou não.

MÓDULO VII

Polimorfismo

Outro ponto essencial na programação orientada a objetos é o chamado polimorfismo. Na natureza, vemos animais que são capazes de alterar sua forma conforme a necessidade, e é dessa ideia que vem o polimorfismo na orientação a objetos. Como sabemos, os objetos filhos herdam as características e ações de seus “ancestrais”. Entretanto, em alguns casos, é necessário que as ações para um mesmo método seja diferente. Em outras palavras, o *polimorfismo* consiste na alteração do funcionamento interno de um método herdado de um objeto pai.

Como um exemplo, temos um objeto genérico “**Eletrodoméstico**”. Esse objeto possui um método, ou ação, “**Ligar()**”. Temos dois objetos, “**Televisão**” e “**Geladeira**”, que não irão ser ligados da mesma forma. Assim, precisamos, para cada uma das classes filhas, reescrever o método “**Ligar()**”.

MÓDULO VII

Com relação ao polimorfismo, valem algumas observações. Como se trata de um assunto que está intimamente conectado à herança, entender os dois juntamente é uma boa ideia. Outro ponto é o fato de que as linguagens de programação implementam o polimorfismo de maneiras diferentes.

O C#, por exemplo, faz uso de método virtuais (com a palavra-chave *virtual*) que podem ser reimplementados (com a palavra-chave *override*) nas classes filhas. Já em Java, apenas o atributo “@Override” é necessário.

Esses quatro pilares são essenciais no entendimento de qualquer linguagem orientada a objetos e da orientação a objetos como um todo. Cada linguagem irá implementar esses pilares de uma forma, mas essencialmente é a mesma coisa. Apenas a questão da herança, como comentado, que pode trazer variações mais bruscas, como a presença de herança múltipla. Além disso, o encapsulamento também é feito de maneiras distintas nas diversas linguagens, embora os *getters* e *setters* sejam praticamente onipresentes.

MÓDULO VII

Principais vantagens da POO

A programação orientada a objetos traz uma ideia muito interessante: a representação de cada elemento em termos de um objeto, ou classe. Esse tipo de representação procura aproximar o sistema que está sendo criado ao que é observado no mundo real, e um objeto contém características e ações, assim como vemos na realidade. Esse tipo de representação traz algumas vantagens muito interessantes para os desenvolvedores e também para o usuário da aplicação. Veremos algumas delas a seguir.

MÓDULO VII

A reutilização de código é um dos principais requisitos no desenvolvimento de software atual.

Com a complexidade dos sistemas cada vez maior, o tempo de desenvolvimento iria aumentar exponencialmente caso não fosse possível a reutilização. A orientação a objetos permite que haja uma reutilização do código criado, diminuindo o tempo de desenvolvimento, bem como o número de linhas de código. Isso é possível devido ao fato de que as linguagens de programação orientada a objetos trazem representações muito claras de cada um dos elementos, e esses elementos normalmente não são interdependentes. Essa independência entre as partes do software é o que permite que esse código seja reutilizado em outros sistemas no futuro.

MÓDULO VII

Outra grande vantagem que o desenvolvimento orientado a objetos traz diz respeito à leitura e manutenção de código. Como a representação do sistema se aproxima muito do que vemos na vida real, o entendimento do sistema como um todo e de cada parte individualmente fica muito mais simples.

Isso permite que a equipe de desenvolvimento não fique dependente de uma pessoa apenas, como acontecia com frequência em linguagens estruturadas como o C, por exemplo.

A criação de bibliotecas é outro ponto que é muito mais simples com a orientação a objetos. No caso das linguagens estruturadas, como o C, temos que as bibliotecas são coleções de procedimentos (ou funções) que podem ser reutilizadas. No caso da **POO**, entretanto, as bibliotecas trazem representações de classes, que são muito mais claras para permitirem a reutilização.

MÓDULO VII

Entretanto, nem tudo é perfeição na programação orientada a objetos. A execução de uma aplicação orientada a objetos é mais lenta do que o que vemos na programação estruturada, por exemplo.

Isso acontece devido à complexidade do modelo, que traz representações na forma de classes. Essas representações irão fazer com que a execução do programa tenha muitos desvios, diferente da execução sequencial da programação estruturada. Esse é o grande motivo por trás da preferência pela linguagem C em hardware limitado, como sistemas embarcados. Também é o motivo pelo qual a programação para sistemas móveis como o Google Android, embora em Java (linguagem orientada a objetos), seja feita o menos orientada a objetos possível.

MÓDULO VII

No momento atual em que estamos, tecnologicamente essa execução mais lenta não é sentida. Isso significa que, em termos de desenvolvimento de sistemas modernos, a programação orientada a objetos é a mais recomendada devido às vantagens que foram apresentadas. Essas vantagens são derivadas do modelo de programação, que busca uma representação baseada no que vemos no mundo real.

MÓDULO VII

Exemplos de Linguagens Orientadas a Objetos

Há uma grande quantidade de linguagens de programação orientada a objetos no mercado atualmente.

Nesse artigo, iremos apresentar 3 das mais utilizadas no momento: **Java**, **C#** e **C++**. Cada uma delas possui uma abordagem diferente do problema que as torna muito boas para alguns tipos de aplicações e não tão boas para outros.

MÓDULO VII

Java

O **Java** é, muito provavelmente, a linguagem de programação mais utilizada no mercado atual. Auxiliado pela presença do **JRE** (*Java Runtime Environment*), ou variações dele, em quase todos os dispositivos eletrônicos do momento, a linguagem **Java** é um grande sucesso entre os desenvolvedores. O sucesso da linguagem aumentou ainda mais com o **Google Android**, que escolheu o **Java** como linguagem preferencial de desenvolvimento de aplicações.

O **Java** implementa os quatro pilares de forma bastante intuitiva, o que facilita o entendimento por parte do desenvolvedor. A abstração, o primeiro pilar, é implementado através de classes, que contém propriedades e métodos, de forma bastante simples. Já o encapsulamento é realizado através de propriedades privadas, auxiliadas por métodos especiais **getters** e **setters**, como mostra a **Listagem 1**. Vale ressaltar a palavra-chave “**this**” mostrada no método **SetId()**. Essa palavra-chave funciona como um representante da classe atual, uma auto-referência ao próprio objeto.

MÓDULO VII

Listagem 1. Encapsulamento em Java

```
private int id;

public int GetId()
{
    return id;
}

public void SetId(int id)
{
    this.id = id;
}
```

As questões de herança e polimorfismo no **Java** são um pouco mais complexas. O **Java** possui herança simples, o que significa que cada classe pode herdar de apenas uma outra. Entretanto, o **Java** possui as chamadas Interfaces, que possuem propriedades e assinaturas de métodos. Essas interfaces precisam ser implementadas para funcionar, o que significa que uma classe pode implementar várias interfaces e herdar de apenas uma classe. Na questão de polimorfismo, o atributo **@Override** é responsável por informar ao **Java** que o método em questão está sendo reescrito.

MÓDULO VII

C#

O C#, por sua vez, é outra das linguagens mais utilizadas no mercado.

Como os computadores pessoais no mundo, em sua maioria, possuem o sistema operacional Windows, da Microsoft, o C# se popularizou. Isso porque o Windows implementa o Framework .NET, ao qual o C# está associado. O C# é uma linguagem de uso geral e especialmente criada para utilização com a orientação a objetos. Vale ressaltar que, em C#, tudo é um objeto (herda da classe **object**).

A abstração é muito simples, e segue o modelo do **Java**. A questão de encapsulamento é um pouco diferente devido a implementação dos métodos **getter** e **setter**. A nomenclatura também é um pouco diferente. A variável que realmente guarda o valor do dado é chamada atributo, enquanto a propriedade é o elemento que realmente acessa aquele dado do mundo externo. Isso está mostrado na **Listagem 2**.

Além disso, o C# faz uso de duas palavras-chave especiais: **get** e **set**.

MÓDULO VII

Listagem 2. Encapsulamento em C#

```
// Atributo
private int id;

// Propriedade
public int Id
{
    get;
    set;
}
```

MÓDULO VII

A questão da herança em **C#** também segue o modelo do **Java**: herança simples e a possibilidade de utilização de interfaces. A importância das interfaces é muito grande, uma vez que elas podem dar o tipo dos dados, que somente posteriormente serão associados a um tipo real, como mostra a **Listagem 3**. Isso também é válido para o **Java**. Por padrão, as identidades das interfaces começam com a letra “I”. O polimorfismo, por sua vez, é baseado em métodos virtuais (com a palavra-chave ***virtual***) na classe pai e reescritos com a palavra-chave ***override*** na classe filha.

```
IExemploInterface exemplo;  
    exemplo = new ImplementacaoIExemploInterface();
```

Listagem 3. Interfaces em C#

MÓDULO VII

C++

O C++, por sua vez, é uma linguagem um pouco mais primitiva, e permite muito mais liberdades com o hardware. Como ele foi derivado imediatamente do C, o C++ permite a utilização de ponteiros, por exemplo, que irão trabalhar diretamente com a memória. Além disso, o C++ pode utilizar todas as bibliotecas C que existem diretamente.

Em termos de abstração, o C++ implementa classes, assim como qualquer linguagem orientada a objetos.

Ele também possui o sentido de privado e público, que é utilizado para encapsulamento.

Esse encapsulamento é realizado através de métodos ***getter*** e ***setter***, muito similar ao visto em **Java**, como mostra a **Listagem 4**. Repare que a listagem mostra somente a assinatura dos métodos especiais, sendo que sua implementação é a mesma que em **Java**. Esse tipo de adaptação é muito comum em C++, onde a classe é guardada em um arquivo **.h** e sua implementação em um arquivo **.cpp**.

MÓDULO VII

Listagem 4. Encapsulamento em C++

```
private:  
    int id;  
public:  
    int GetId() const;  
    void SetId(int const id);
```

A questão da herança no C++ é um pouco diferente. A linguagem permite a herança múltipla, o que significa que cada classe pode herdar de quantas classes desejar. Isso pode causar problemas de métodos que possuem o mesmo nome, portanto o desenvolvedor precisa estar atento. O polimorfismo é baseado em métodos virtuais, da mesma forma como o C#. A complexidade, entretanto, é maior, uma vez que temos que cuidar de detalhes de mais baixo nível, como acesso a memória.

MÓDULO VII

Além dessas exemplificadas, existem outras linguagens que merecem ser citadas.

Entre elas, podemos elencar: **Python**, linguagem de script orientada a objetos que é muito utilizada em pesquisas científicas devido a sua velocidade; **Object Pascal** (também conhecida como **Delphi**, devido ao nome de sua **IDE**), apesar do grande número de sistemas mais antigos que a utilizam; **Objective-C**, que é a linguagem de preferência para desenvolvimento de aplicações para os sistemas da **Apple**, como **iPhone** e **iPad**; **Ruby**, voltada para o desenvolvimento web; e **Visual Basic .NET**, muito utilizada até pouco tempo, mas também caindo em desuso, principalmente devido ao avanço do **C#** em popularidade.

MÓDULO VII

EXERCÍCIOS...

MÓDULO VII

FIM...