Fundamentos do PHP e Estruturas de Decisão para Desenvolvedores

1. Introdução ao Ecossistema PHP

O PHP é uma linguagem de programação de propósito geral que carrega em seu cerne uma especialização profunda para o desenvolvimento web. Para estudantes que já dominam a lógica de programação em pseudocódigo, o PHP representa o passo estratégico ideal: é a transição da abstração lógica para a construção de sistemas reais que operam sob o **protocolo HTTP**. No modelo **Cliente-Servidor**, o PHP atua no **Back-end**, processando a lógica no servidor (como no caso do Instagram, onde o PHP gerencia quem deve ver qual foto) antes de enviar uma resposta ao navegador do cliente.

Embora frequentemente classificado como uma linguagem interpretada, o PHP moderno é extremamente performático. Ele utiliza um processo de **compilação interna** (gerando **Opcodes**) que ocorre em segundo plano. Isso significa que o desenvolvedor desfruta da agilidade de ver as alterações instantaneamente, sem a burocracia de um ciclo de compilação manual, enquanto o motor da linguagem garante a eficiência na execução.

2. Anatomia de um Script PHP: Da Teoria à Prática

Um arquivo PHP é essencialmente um script de texto com a extensão `.php`. É importante notar que, embora o interpretador PHP possa executar arquivos sem essa extensão em certos contextos de terminal, ela é indispensável para que **editores como o VS Code** e servidores web identifiquem a linguagem e apliquem o realce de sintaxe e as ferramentas de automação corretamente.

A estrutura de um arquivo profissional, como o nosso projeto **Screen Match**, segue regras rígidas:

- **Tag de Abertura:** O uso de `<?php` é obrigatório para indicar que o conteúdo a seguir deve ser processado pelo motor do PHP.
- **Finalização de Instruções:** Cada comando deve ser encerrado com ponto e vírgula `;`. O PHP é rigoroso quanto a isso; a omissão resulta em um erro fatal.
- **Saída de Dados:** O comando `echo` é o responsável por enviar informações para a saída padrão.

```php
<?php

// Exemplo inicial no projeto Screen Match

echo "Bem-vindo(a) ao Screen Match!\n";
```

A sintaxe do PHP permite uma organização visual livre (espaços e quebras de linha), o que favorece a legibilidade. Contudo, essa liberdade exige disciplina do desenvolvedor para manter o código limpo, pois o rigor sintático do ponto e vírgula não perdoa distrações.

3. Gerenciamento de Dados e Tipagem Dinâmica

A transição do pseudocódigo para o PHP substitui os comandos genéricos por uma sintaxe de atribuição clara. No PHP, as variáveis são identificadas pelo prefixo **`$`** e seguem a convenção **`camelCase`** (ex: `$nomeFilme`).

A linguagem é **dinamicamente tipada**, o que significa que o tipo da variável é definido pelo valor atribuído a ela:

- **string**: Textos envoltos em aspas.
- **integer**: Números inteiros.
- **float / double**: Números decimais (no PHP, ambos são essencialmente o mesmo tipo).
- **boolean**: Valores lógicos (`true` ou `false`).
- **null**: Representa a ausência deliberada de valor.

Um diferencial pragmático é o suporte a **"Strings Numéricas"**: o PHP consegue realizar operações matemáticas se uma string contiver apenas números. Por exemplo, `"5" + "10"` resultará no inteiro `15`.

A tipagem dinâmica oferece uma agilidade tremenda na escrita do código, eliminando a necessidade de declarações burocráticas. Entretanto, essa facilidade exige maior responsabilidade na nomeação das variáveis para garantir que a intenção do código permaneça clara durante a manutenção a longo prazo.

4. Interatividade: Entrada de Dados via Terminal

No pseudocódigo, utilizávamos o comando "LEIA" para capturar dados. No PHP, quando operamos via terminal (CLI), essa função é desempenhada pela variável global **`$argv`**. Ela é um **array** (lista) que armazena os argumentos passados na execução:

- `$argv[0]`: Sempre contém o nome do script executado.
- `$argv[1]`: Captura o primeiro parâmetro informado pelo usuário.

Para garantir a estabilidade do sistema, utilizamos o **operador de coalescência nula (`??`)**, uma prática de **programação defensiva**:

```
// Capturando o ano de lançamento ou definindo 2022 como padrão

$anoLancamento = $argv[1] ?? 2022;

echo "Ano de lançamento: $anoLancamento";
```

Tratar a entrada de dados logo na origem evita que o programa tente processar valores nulos, o que causaria falhas. É a base para criar softwares resilientes a falhas humanas.

5. O Coração da Lógica: Operadores Booleanos

A lógica de decisão no PHP é construída sobre expressões que resultam em um valor `boolean`. Dominar esses operadores é o que permite traduzir regras de negócio em código funcional.

- **Comparação:** `==` (igualdade), `>` (maior que), `<` (menor que), `>=` (maior ou igual), `<=` (menor ou igual).
- **Lógicos:** `&&` (AND - todas as condições verdadeiras), `||` (OR - pelo menos uma verdadeira), `!` (NOT - inversão lógica).

Estas expressões formam a inteligência do software. No **Screen Match**, é através dessa lógica que decidimos, por exemplo, se um filme será recomendado com base na combinação de sua nota e ano de lançamento.

6. Estruturas de Controle e Tomada de Decisão

O PHP oferece ferramentas distintas para desviar o fluxo do programa, cada uma com seu propósito arquitetural:

I. Bloco If / Elseif / Else (Statements)

O `if` é uma **instrução (statement)**. Ele executa blocos de código com base em condições complexas, mas não retorna um valor por si só. É ideal para encadeamentos lógicos que exigem avaliações de intervalos (ex: `$nota > 5 && $nota < 8`).

II. Switch-Case

Indicado para múltiplas verificações de **igualdade simples** contra uma mesma variável. É mais legível que diversos `if` aninhados, mas exige o uso do `break` para interromper a execução e um `default` para casos não previstos.

III. Match Expression (A Evolução Moderna)

Diferente do `if` e do `switch`, o `match` é uma **expressão**. Isso significa que ele **retorna um valor**, permitindo atribuição direta. Além disso, o `match` utiliza **comparação estrita (`===`)**, o que evita comportamentos inesperados de conversão de tipos, e exige que todos os casos sejam tratados (ou que um `default` seja definido).

```
$genero = match ($nomeFilme) {
    "Top Gun - Maverick" => "Ação",
    "Thor: Ragnarok"     => "Super-herói",
    "Se Beber Não Case"  => "Comédia",
    default              => "Gênero desconhecido",
};
```

A escolha entre `switch` e `match` é uma questão de segurança e concisão. O `match` é a escolha superior no PHP moderno por ser mais seguro (strict comparison) e forçar o desenvolvedor a lidar com todos os cenários possíveis, reduzindo bugs latentes.