Table of Contents

# Guia Completo de JavaScript, TypeScript e PHP

Este guia tem como objetivo fornecer um material de consulta completo e organizado sobre operadores, estruturas de controle e programação orientada a objetos nas linguagens JavaScript, TypeScript e PHP. O foco é apresentar os conceitos de forma integrada, destacando as semelhanças e diferenças de sintaxe e comportamento entre as linguagens, sempre com base nas versões mais recentes e estáveis.

## 1. Introdução

### Sobre este guia

Este guia foi elaborado para desenvolvedores que trabalham com MySQL, Laravel e React Native, e buscam um material de consulta rápido e eficiente para revisar os fundamentos de JavaScript, TypeScript e PHP. A abordagem é focada na integração e comparação entre as linguagens, facilitando a compreensão das semelhanças e diferenças sintáticas e conceituais. O conteúdo é organizado por capítulos, com links diretos para a documentação oficial, garantindo a precisão e a atualização das informações.

### Versões das linguagens

Para garantir a relevância e a precisão das informações, este guia se baseia nas versões mais recentes e estáveis das linguagens no momento de sua criação:

* JavaScript (Node.js LTS): v22 (LTS - “Jod”)
* TypeScript: 5.9.2
* PHP: 8.3 (versão estável atual), com menções à 8.4 (futura estável) quando relevante.

### Documentação oficial

Para aprofundamento e consulta detalhada, recomenda-se sempre a documentação oficial das respectivas linguagens:

* JavaScript: [MDN Web Docs - JavaScript](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript)
* TypeScript: [TypeScript Documentation](https://www.typescriptlang.org/docs/)
* PHP: [Manual do PHP](https://www.php.net/manual/pt_BR/)

## 2. Operadores

Operadores são símbolos que instruem o interpretador a realizar alguma operação matemática ou lógica. Eles são fundamentais em qualquer linguagem de programação para manipular valores e controlar o fluxo de execução. Embora JavaScript, TypeScript e PHP compartilhem muitos operadores em comum devido às suas raízes e influências, existem diferenças sutis e específicas de cada linguagem que serão abordadas nesta seção.

### 2.1. Operadores Aritméticos

Utilizados para realizar operações matemáticas básicas.

| Operador | Descrição | Exemplo (JS/TS) | Exemplo (PHP) |
| --- | --- | --- | --- |
| + | Adição | 5 + 3 | 5 + 3 |
| - | Subtração | 5 - 3 | 5 - 3 |
| \* | Multiplicação | 5 \* 3 | 5 \* 3 |
| / | Divisão | 5 / 3 | 5 / 3 |
| % | Módulo (Resto) | 5 % 3 | 5 % 3 |
| \*\* | Exponenciação | 5 \*\* 3 | 5 \*\* 3 |

**JavaScript/TypeScript:**

**JavaScript:**

let a = 10;  
let b = 3;  
console.log(a + b); // 13  
console.log(a - b); // 7  
console.log(a \* b); // 30  
console.log(a / b); // 3.333...  
console.log(a % b); // 1  
console.log(a \*\* b); // 1000 (10 elevado à potência de 3)

**TypeScript:**

let a: number = 10;  
let b: number = 3;  
console.log(a + b); // 13  
console.log(a - b); // 7  
console.log(a \* b); // 30  
console.log(a / b); // 3.333...  
console.log(a % b); // 1  
console.log(a \*\* b); // 1000 (10 elevado à potência de 3)

**PHP:**

$a = 10;  
$b = 3;  
echo $a + $b; // 13  
echo $a - $b; // 7  
echo $a \* $b; // 30  
echo $a / $b; // 3.333...  
echo $a % $b; // 1  
echo $a \*\* $b; // 1000 (10 elevado à potência de 3)

**Observações:**

* Todos os operadores aritméticos funcionam de forma similar nas três linguagens.
* Em JavaScript/TypeScript, a divisão por zero resulta em Infinity ou NaN (Not-a-Number), dependendo do operando. Em PHP, a divisão por zero gera um aviso (Warning) e o resultado é INF (Infinity) ou NAN.

**Documentação Oficial:**

* JavaScript: [Operadores Aritméticos](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/Arithmetic_operators)
* PHP: [Operadores Aritméticos](https://www.php.net/manual/pt_BR/language.operators.arithmetic.php)

### 2.2. Operadores de Atribuição

Utilizados para atribuir valores a variáveis.

| Operador | Exemplo (JS/TS) | Equivalente (JS/TS) | Exemplo (PHP) | Equivalente (PHP) |
| --- | --- | --- | --- | --- |
| = | x = 5 | x = 5 | $x = 5 | $x = 5 |
| += | x += 5 | x = x + 5 | $x += 5 | $x = $x + 5 |
| -= | x -= 5 | x = x - 5 | $x -= 5 | $x = $x - 5 |
| \*= | x \*= 5 | x = x \* 5 | $x \*= 5 | $x = $x \* 5 |
| /= | x /= 5 | x = x / 5 | $x /= 5 | $x = $x / 5 |
| %= | x %= 5 | x = x % 5 | $x %= 5 | $x = $x % 5 |
| \*\*= | x \*\*= 5 | x = x \*\* 5 | $x \*\*= 5 | $x = $x \*\* 5 |

**JavaScript/TypeScript:**

**JavaScript:**

let x = 10;  
x += 5; // x agora é 15  
console.log(x);  
x -= 3; // x agora é 12  
console.log(x);  
x \*= 2; // x agora é 24  
console.log(x);  
x /= 4; // x agora é 6  
console.log(x);  
x %= 5; // x agora é 1  
console.log(x);  
x \*\*= 3; // x agora é 1 (1 elevado à potência de 3)  
console.log(x);

**TypeScript:**

let x: number = 10;  
x += 5; // x agora é 15  
console.log(x);  
x -= 3; // x agora é 12  
console.log(x);  
x \*= 2; // x agora é 24  
console.log(x);  
x /= 4; // x agora é 6  
console.log(x);  
x %= 5; // x agora é 1  
console.log(x);  
x \*\*= 3; // x agora é 1 (1 elevado à potência de 3)  
console.log(x);

**PHP:**

$x = 10;  
$x += 5; // $x agora é 15  
echo $x . "\n";  
$x -= 3; // $x agora é 12  
echo $x . "\n";  
$x \*= 2; // $x agora é 24  
echo $x . "\n";  
$x /= 4; // $x agora é 6  
echo $x . "\n";  
$x %= 5; // $x agora é 1  
echo $x . "\n";  
$x \*\*= 3; // $x agora é 1 (1 elevado à potência de 3)  
echo $x . "\n";

**Observações:**

* Os operadores de atribuição compostos (+=, -=, etc.) funcionam de maneira idêntica nas três linguagens.

**Documentação Oficial:**

* JavaScript: [Operadores de Atribuição](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/Assignment_operators)
* PHP: [Operadores de Atribuição](https://www.php.net/manual/pt_BR/language.operators.assignment.php)

### 2.3. Operadores de Comparação

Utilizados para comparar dois valores e retornar um valor booleano (true ou false).

| Operador | Descrição | Exemplo (JS/TS) | Exemplo (PHP) |
| --- | --- | --- | --- |
| == | Igual a (com conversão de tipo) | 5 == '5' | 5 == '5' |
| === | Estritamente igual a (sem conversão) | 5 === '5' | 5 === '5' |
| != | Diferente de (com conversão de tipo) | 5 != '5' | 5 != '5' |
| !== | Estritamente diferente de (sem conversão) | 5 !== '5' | 5 !== '5' |
| > | Maior que | 5 > 3 | 5 > 3 |
| < | Menor que | 5 < 3 | 5 < 3 |
| >= | Maior ou igual a | 5 >= 3 | 5 >= 3 |
| <= | Menor ou igual a | 5 <= 3 | 5 <= 3 |
| <=> | Spaceship (PHP 7+) | N/A | 1 <=> 2 |

**JavaScript/TypeScript:**

**JavaScript:**

console.log(5 == '5'); // true (coerção de tipo)  
console.log(5 === '5'); // false (tipos diferentes)  
console.log(5 != '5'); // false  
console.log(5 !== '5'); // true  
console.log(10 > 5); // true  
console.log(10 < 5); // false  
console.log(10 >= 10); // true  
console.log(10 <= 5); // false

**TypeScript:**

// TypeScript permite a comparação, mas o linter/compilador pode alertar sobre comparações de tipos diferentes  
console.log(5 == '5'); // true (coerção de tipo)  
console.log(5 === '5'); // false (tipos diferentes)  
console.log(5 != '5'); // false  
console.log(5 !== '5'); // true  
console.log(10 > 5); // true  
console.log(10 < 5); // false  
console.log(10 >= 10); // true  
console.log(10 <= 5); // false  
  
// Exemplo com tipagem explícita para clareza  
let num1: number = 5;  
let str1: string = '5';  
  
// console.log(num1 == str1); // Erro em tempo de compilação se strictEqualityChecks estiver ativado  
console.log(num1 === Number(str1)); // true (conversão explícita)

**PHP:**

var\_dump(5 == '5'); // bool(true)  
var\_dump(5 === '5'); // bool(false)  
var\_dump(5 != '5'); // bool(false)  
var\_dump(5 !== '5'); // bool(true)  
var\_dump(10 > 5); // bool(true)  
var\_dump(10 < 5); // bool(false)  
var\_dump(10 >= 10); // bool(true)  
var\_dump(10 <= 5); // bool(false)  
  
// Operador Spaceship (PHP 7+)  
var\_dump(1 <=> 1); // int(0) (igual)  
var\_dump(1 <=> 2); // int(-1) (esquerda é menor)  
var\_dump(2 <=> 1); // int(1) (esquerda é maior)

**Observações:**

* **Coerção de Tipo:** JavaScript e PHP realizam coerção de tipo com == e !=. Isso significa que eles tentam converter os operandos para um tipo comum antes da comparação. TypeScript, por ser um superconjunto de JavaScript, herda esse comportamento, mas o uso de tipos estáticos em TypeScript geralmente ajuda a evitar comparações que dependem de coerção implícita.
* **Comparação Estrita (===, !==):** É altamente recomendado usar os operadores de comparação estrita (=== e !==) em JavaScript, TypeScript e PHP, pois eles comparam tanto o valor quanto o tipo, evitando comportamentos inesperados devido à coerção de tipo.
* **Operador Spaceship (<=>):** Exclusivo do PHP (a partir da versão 7). Ele retorna 0 se os operandos forem iguais, -1 se o operando da esquerda for menor, e 1 se o operando da esquerda for maior. É útil para funções de comparação (callbacks) em ordenação.

**Documentação Oficial:**

* JavaScript: [Operadores de Comparação](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/Comparison_operators)
* PHP: [Operadores de Comparação](https://www.php.net/manual/pt_BR/language.operators.comparison.php)

### 2.4. Operadores Lógicos

Utilizados para combinar ou negar expressões booleanas.

| Operador | Descrição | Exemplo (JS/TS) | Exemplo (PHP) |
| --- | --- | --- | --- |
| && | AND lógico | a && b | $a && $b |
|  |  |  | OR lógico |

**JavaScript/TypeScript:**

let idade = 20;  
let temHabilitacao = true;  
console.log(idade >= 18 && temHabilitacao); // true (ambas as condições são verdadeiras)  
console.log(idade < 18 || temHabilitacao); // true (pelo menos uma condição é verdadeira)  
console.log(!temHabilitacao); // false (negação de true)

**TypeScript:**

let idade: number = 20;  
let temHabilitacao: boolean = true;  
console.log(idade >= 18 && temHabilitacao); // true (ambas as condições são verdadeiras)  
console.log(idade < 18 || temHabilitacao); // true (pelo menos uma condição é verdadeira)  
console.log(!temHabilitacao); // false (negação de true)

**PHP:**

$idade = 20;  
$temHabilitacao = true;  
var\_dump($idade >= 18 && $temHabilitacao); // bool(true)  
var\_dump($idade < 18 || $temHabilitacao); // bool(true)  
var\_dump(!$temHabilitacao); // bool(false)

**Observações:**

* Os operadores lógicos (&&, ||, !) funcionam de forma idêntica nas três linguagens, incluindo o comportamento de curto-circuito (short-circuit evaluation), onde a segunda expressão só é avaliada se a primeira não for suficiente para determinar o resultado final.

**Documentação Oficial:**

* JavaScript: [Operadores Lógicos](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/Logical_AND)
* PHP: [Operadores Lógicos](https://www.php.net/manual/pt_BR/language.operators.logical.php)

### 2.5. Operadores de String

Utilizados para manipular strings.

| Operador | Descrição | Exemplo (JS/TS) | Exemplo (PHP) |
| --- | --- | --- | --- |
| + | Concatenação (JS/TS) | 'Olá' + ' Mundo' | N/A |
| . | Concatenação (PHP) | N/A | 'Olá' . ' Mundo' |
| += | Atribuição com Concatenação (JS/TS) | str += '!' | N/A |
| .= | Atribuição com Concatenação (PHP) | N/A | $str .= '!' |

**JavaScript/TypeScript:**

let saudacao = 'Olá';  
let nome = 'Mundo';  
let mensagem = saudacao + ' ' + nome; // Olá Mundo  
console.log(mensagem);  
saudacao += '!'; // Olá!  
console.log(saudacao);

**TypeScript:**

let saudacao: string = 'Olá';  
let nome: string = 'Mundo';  
let mensagem: string = saudacao + ' ' + nome; // Olá Mundo  
console.log(mensagem);  
saudacao += '!'; // Olá!  
console.log(saudacao);

**PHP:**

$saudacao = 'Olá';  
$nome = 'Mundo';  
$mensagem = $saudacao . ' ' . $nome; // Olá Mundo  
echo $mensagem . "\n";  
$saudacao .= '!'; // Olá!  
echo $saudacao . "\n";

**Observações:**

* **JavaScript/TypeScript:** Utilizam o operador + para concatenação de strings. Se um dos operandos for uma string e o outro não, o outro operando será convertido para string antes da concatenação.
* **PHP:** Utiliza o operador . para concatenação de strings. O operador + em PHP é estritamente para operações aritméticas.

**Documentação Oficial:**

* JavaScript: [Operador de Concatenação](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/Addition)
* PHP: [Operadores de String](https://www.php.net/manual/pt_BR/language.operators.string.php)

### 2.6. Operadores de Incremento/Decremento

Utilizados para aumentar ou diminuir o valor de uma variável em uma unidade.

| Operador | Descrição | Exemplo (JS/TS) | Exemplo (PHP) |
| --- | --- | --- | --- |
| ++ | Incremento | x++ ou ++x | $x++ ou ++$x |
| -- | Decremento | x-- ou --x | $x-- ou --$x |

**JavaScript/TypeScript:**

let contador = 0;  
console.log(contador++); // 0 (pós-incremento: usa o valor atual, depois incrementa)  
console.log(contador); // 1  
console.log(++contador); // 2 (pré-incremento: incrementa, depois usa o novo valor)  
console.log(contador); // 2  
  
let valor = 10;  
console.log(valor--); // 10 (pós-decremento: usa o valor atual, depois decrementa)  
console.log(valor); // 9  
console.log(--valor); // 8 (pré-decremento: decrementa, depois usa o novo valor)  
console.log(valor); // 8

**TypeScript:**

let contador: number = 0;  
console.log(contador++); // 0 (pós-incremento: usa o valor atual, depois incrementa)  
console.log(contador); // 1  
console.log(++contador); // 2 (pré-incremento: incrementa, depois usa o novo valor)  
console.log(contador); // 2  
  
let valor: number = 10;  
console.log(valor--); // 10 (pós-decremento: usa o valor atual, depois decrementa)  
console.log(valor); // 9  
console.log(--valor); // 8 (pré-decremento: decrementa, depois usa o novo valor)  
console.log(valor); // 8

**PHP:**

$contador = 0;  
echo $contador++ . "\n"; // 0 (pós-incremento)  
echo $contador . "\n"; // 1  
echo ++$contador . "\n"; // 2 (pré-incremento)  
echo $contador . "\n"; // 2  
  
$valor = 10;  
echo $valor-- . "\n"; // 10 (pós-decremento)  
echo $valor . "\n"; // 9  
echo --$valor . "\n"; // 8 (pré-decremento)  
echo $valor . "\n"; // 8

**Observações:**

* O comportamento de pré-incremento/decremento e pós-incremento/decremento é idêntico nas três linguagens.

**Documentação Oficial:**

* JavaScript: [Operadores de Incremento e Decremento](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/Increment)
* PHP: [Operadores de Incremento/Decremento](https://www.php.net/manual/pt_BR/language.operators.increment.php)

### 2.7. Outros Operadores

#### Operador Ternário (Condicional)

Um atalho para a instrução if...else.

| Operador | Descrição | Sintaxe | Exemplo (JS/TS) | Exemplo (PHP) |
| --- | --- | --- | --- | --- |
| ? : | Ternário | condicao ? expressao1 : expressao2 | idade >= 18 ? 'Maior' : 'Menor' | $idade >= 18 ? 'Maior' : 'Menor' |

**JavaScript/TypeScript:**

let idade = 20;  
let status = (idade >= 18) ? 'Maior de idade' : 'Menor de idade';  
console.log(status); // Maior de idade

**TypeScript:**

let idade: number = 20;  
let status: string = (idade >= 18) ? 'Maior de idade' : 'Menor de idade';  
console.log(status); // Maior de idade

**PHP:**

$idade = 20;  
$status = ($idade >= 18) ? 'Maior de idade' : 'Menor de idade';  
echo $status . "\n"; // Maior de idade

**Observações:**

* O operador ternário funciona de forma idêntica nas três linguagens, oferecendo uma forma concisa de expressar condicionais simples.

**Documentação Oficial:**

* JavaScript: [Operador Condicional (Ternário)](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/Conditional_operator)
* PHP: [Operador Ternário](https://www.php.net/manual/pt_BR/language.operators.comparison.php#language.operators.comparison.ternary)

#### Operador typeof (JavaScript/TypeScript)

Retorna uma string indicando o tipo do operando.

**JavaScript/TypeScript:**

console.log(typeof 42); // "number"  
console.log(typeof "hello"); // "string"  
console.log(typeof true); // "boolean"  
console.log(typeof undefined); // "undefined"  
console.log(typeof null); // "object" (um erro histórico do JavaScript)  
console.log(typeof {}); // "object"  
console.log(typeof []); // "object"  
console.log(typeof function(){}); // "function"

**TypeScript:**

console.log(typeof 42); // "number"  
console.log(typeof "hello"); // "string"  
console.log(typeof true); // "boolean"  
console.log(typeof undefined); // "undefined"  
console.log(typeof null); // "object" (um erro histórico do JavaScript)  
console.log(typeof {}); // "object"  
console.log(typeof []); // "object"  
console.log(typeof function(){}); // "function"  
  
// Em TypeScript, a tipagem estática permite que você saiba o tipo de uma variável  
// antes mesmo de usar `typeof` em tempo de execução.  
let num: number = 42;  
let str: string = "hello";  
let bool: boolean = true;  
let undef: undefined = undefined;  
let nulo: null = null;  
let obj: object = {};  
let arr: any[] = [];  
let func: Function = function(){};  
  
// O uso de `typeof` em TypeScript é mais para inspeção em tempo de execução,  
// já que o compilador já conhece os tipos.  
console.log(`Tipo de num: ${typeof num}`);  
console.log(`Tipo de str: ${typeof str}`);  
console.log(`Tipo de bool: ${typeof bool}`);  
console.log(`Tipo de undef: ${typeof undef}`);  
console.log(`Tipo de nulo: ${typeof nulo}`);  
console.log(`Tipo de obj: ${typeof obj}`);  
console.log(`Tipo de arr: ${typeof arr}`);  
console.log(`Tipo de func: ${typeof func}`);

**Observações:**

* O operador typeof é útil para verificar tipos primitivos. Para objetos, ele retorna "object" (exceto para funções). Para null, ele retorna "object", o que é uma peculiaridade histórica do JavaScript.

**Documentação Oficial:**

* [JavaScript: Operador typeof](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/typeof)

#### Operador instanceof (JavaScript/TypeScript)

Verifica se um objeto é uma instância de uma classe ou de um tipo de objeto específico.

**JavaScript/TypeScript:**

class Animal {  
 constructor(nome) {  
 this.nome = nome;  
 }  
}  
  
class Cachorro extends Animal {  
 constructor(nome, raca) {  
 super(nome);  
 this.raca = raca;  
 }  
}  
  
let meuCachorro = new Cachorro("Rex", "Labrador");  
let meuAnimal = new Animal("Leão");  
  
console.log(meuCachorro instanceof Cachorro); // true  
console.log(meuCachorro instanceof Animal); // true  
console.log(meuAnimal instanceof Cachorro); // false  
console.log([] instanceof Array); // true  
console.log({} instanceof Object); // true

**TypeScript:**

class AnimalTS {  
 constructor(public nome: string) {}  
}  
  
class CachorroTS extends AnimalTS {  
 constructor(nome: string, public raca: string) {  
 super(nome);  
 }  
}  
  
let meuCachorroTS: CachorroTS = new CachorroTS("Rex", "Labrador");  
let meuAnimalTS: AnimalTS = new AnimalTS("Leão");  
  
console.log(meuCachorroTS instanceof CachorroTS); // true  
console.log(meuCachorroTS instanceof AnimalTS); // true  
console.log(meuAnimalTS instanceof CachorroTS); // false  
console.log([] instanceof Array); // true  
console.log({} instanceof Object); // true  
  
// Em TypeScript, você pode usar a tipagem para garantir que um objeto  
// é de um determinado tipo, o que é verificado em tempo de compilação.  
function processarAnimal(animal: AnimalTS) {  
 if (animal instanceof CachorroTS) {  
 console.log(`É um cachorro da raça: ${animal.raca}`);  
 } else {  
 console.log(`É um animal genérico: ${animal.nome}`);  
 }  
}  
  
processarAnimal(meuCachorroTS);  
processarAnimal(meuAnimalTS);

**Observações:**

* O operador instanceof verifica a cadeia de protótipos do objeto. Ele retorna true se o protótipo do construtor estiver presente na cadeia de protótipos do objeto.

**Documentação Oficial:**

* JavaScript: [Operador instanceof](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/instanceof)

#### Funções de Verificação de Tipo em PHP

PHP não possui operadores typeof ou instanceof no mesmo sentido que JavaScript/TypeScript. Em vez disso, ele oferece uma série de funções para verificar o tipo de uma variável e o operador instanceof para verificar a instância de uma classe.

**Funções de Verificação de Tipo:**

* gettype(): Retorna o tipo da variável como uma string.
* is\_array(): Verifica se a variável é um array.
* is\_bool(): Verifica se a variável é um booleano.
* is\_callable(): Verifica se o conteúdo da variável pode ser chamado como uma função.
* is\_float(): Verifica se a variável é um float (número de ponto flutuante).
* is\_int(): Verifica se a variável é um inteiro.
* is\_null(): Verifica se a variável é NULL.
* is\_numeric(): Verifica se a variável é um número ou uma string numérica.
* is\_object(): Verifica se a variável é um objeto.
* is\_resource(): Verifica se a variável é um recurso.
* is\_string(): Verifica se a variável é uma string.

**PHP:**

$numero = 123;  
$texto = "Olá";  
$booleano = true;  
$array = [1, 2, 3];  
$objeto = new stdClass();  
  
echo gettype($numero) . "\n"; // integer  
echo gettype($texto) . "\n"; // string  
echo gettype($booleano) . "\n"; // boolean  
echo gettype($array) . "\n"; // array  
echo gettype($objeto) . "\n"; // object  
  
var\_dump(is\_int($numero)); // bool(true)  
var\_dump(is\_string($texto)); // bool(true)  
var\_dump(is\_array($array)); // bool(true)  
var\_dump(is\_object($objeto)); // bool(true)

**Operador instanceof**\*\* (PHP):\*\*

class Animal {}  
class Cachorro extends Animal {}  
  
$meuCachorro = new Cachorro();  
$meuAnimal = new Animal();  
  
var\_dump($meuCachorro instanceof Cachorro); // bool(true)  
var\_dump($meuCachorro instanceof Animal); // bool(true)  
var\_dump($meuAnimal instanceof Cachorro); // bool(false)

**Observações:**

* Em PHP, gettype() retorna o tipo da variável como uma string, enquanto as funções is\_...() retornam um booleano. O operador instanceof em PHP funciona de forma similar ao JavaScript/TypeScript para verificar a instância de uma classe.

**Documentação Oficial:**

* PHP: [Funções de Tipo](https://www.php.net/manual/pt_BR/ref.var.php)
* PHP: [Operador instanceof](https://www.php.net/manual/pt_BR/language.operators.type.php)

### 2.8. Diferença de Tipagem: JavaScript vs. TypeScript

Uma das distinções mais fundamentais entre JavaScript e TypeScript reside em seus sistemas de tipagem. Compreender essa diferença é crucial para escrever código robusto e escalável, especialmente em projetos maiores.

#### JavaScript: Tipagem Dinâmica e Fraca

JavaScript é uma linguagem de **tipagem dinâmica** e **fraca**.

* **Tipagem Dinâmica (Dynamic Typing):** O tipo de uma variável é determinado em tempo de execução, não em tempo de compilação. Isso significa que você pode reatribuir uma variável com um valor de um tipo diferente sem gerar um erro.
* **Tipagem Fraca (Weak Typing / Loose Typing):** JavaScript permite conversões de tipo implícitas (coerção de tipo) em muitas operações. Isso pode levar a comportamentos inesperados se você não estiver ciente das regras de coerção.

**Vantagens da Tipagem Dinâmica/Fraca (JavaScript):**

* **Flexibilidade:** Permite prototipagem rápida e código mais conciso.
* **Menos verbosidade:** Não é necessário declarar tipos explicitamente.

**Desvantagens da Tipagem Dinâmica/Fraca (JavaScript):**

* **Erros em tempo de execução:** Muitos erros relacionados a tipos só são descobertos quando o código é executado, o que pode dificultar a depuração.
* **Manutenção:** Em projetos grandes, a falta de informações de tipo pode tornar o código mais difícil de entender, manter e refatorar.
* **Previsibilidade:** O comportamento de coerção de tipo pode ser confuso e levar a bugs sutis.

#### TypeScript: Tipagem Estática Opcional

TypeScript é um superconjunto de JavaScript que adiciona **tipagem estática opcional**.

* **Tipagem Estática (Static Typing):** Os tipos das variáveis são verificados em tempo de compilação (antes da execução do código). Isso significa que o compilador TypeScript pode identificar muitos erros relacionados a tipos antes mesmo de o código ser executado.
* **Tipagem Opcional:** Você não é obrigado a usar tipos em todas as variáveis. TypeScript pode inferir tipos automaticamente (inferência de tipo) se você não os declarar explicitamente. Isso permite uma transição gradual de JavaScript para TypeScript.
* **Tipagem Forte (Strong Typing):** Embora TypeScript compile para JavaScript (que é fracamente tipado), ele impõe regras de tipo mais rigorosas em tempo de compilação, reduzindo a chance de coerções de tipo inesperadas.

**Vantagens da Tipagem Estática (TypeScript):**

* **Detecção precoce de erros:** Muitos bugs são pegos durante o desenvolvimento, antes que o código chegue à produção.
* **Melhor legibilidade e manutenção:** O código com tipos explícitos é mais fácil de entender, especialmente em equipes e projetos grandes.
* **Refatoração segura:** O compilador ajuda a garantir que as refatorações não quebrem o código.
* **Melhor autocompletar e ferramentas de desenvolvimento:** IDEs e editores de código podem fornecer sugestões mais precisas e navegação de código aprimorada.
* **Documentação implícita:** Os tipos servem como uma forma de documentação para o código.

**Desvantagens da Tipagem Estática (TypeScript):**

* **Curva de aprendizado:** Requer um tempo para aprender a sintaxe e os conceitos de tipagem.
* **Mais verbosidade:** Pode tornar o código um pouco mais longo devido às declarações de tipo.
* **Etapa de compilação:** Adiciona uma etapa de compilação ao fluxo de trabalho de desenvolvimento.

#### Comparativo Direto

| Característica | JavaScript (JS) | TypeScript (TS) |
| --- | --- | --- |
| **Sistema de Tipos** | Dinâmico e Fraco | Estático (opcional) e Forte (em compilação) |
| **Verificação** | Em tempo de execução | Em tempo de compilação |
| **Detecção de Erros** | Tardia (runtime) | Precoce (compile-time) |
| **Coerção de Tipo** | Implícita (frequente) | Explícita (geralmente exigida) |
| **Curva de Aprendizado** | Baixa (para iniciantes) | Média (adiciona conceitos de tipo) |
| **Produtividade** | Rápida para protótipos pequenos | Alta para projetos grandes e complexos |
| **Ferramentas (IDE)** | Básica | Avançada (autocompletar, refatoração segura) |
| **Compatibilidade** | Executa diretamente no navegador/Node.js | Requer compilação para JavaScript |

Em resumo, enquanto JavaScript oferece flexibilidade e rapidez para pequenos scripts, TypeScript brilha em projetos maiores e mais complexos, onde a segurança de tipo e a manutenibilidade são cruciais. A escolha entre eles depende do tamanho do projeto, da equipe e dos requisitos de robustez.

**Documentação Oficial:**

* TypeScript: [Handbook - Everyday Types](https://www.typescriptlang.org/docs/handbook/2/everyday-types.html)
* TypeScript: [Handbook - Type Inference](https://www.typescriptlang.org/docs/handbook/type-inference.html)

## 3. Estruturas de Controle

As estruturas de controle são blocos de código que permitem controlar o fluxo de execução de um programa, baseando-se em condições ou na repetição de tarefas. JavaScript, TypeScript e PHP compartilham muitas dessas estruturas, com pequenas variações sintáticas e de comportamento.

### 3.1. Estruturas Condicionais

#### if, else if, else

Permitem que o código seja executado condicionalmente.

**Sintaxe Geral:**

if (condicao) {  
 // código a ser executado se a condição for verdadeira  
} else if (outraCondicao) {  
 // código a ser executado se a outraCondicao for verdadeira  
} else {  
 // código a ser executado se nenhuma das condições anteriores for verdadeira  
}

**JavaScript/TypeScript:**

**JavaScript:**

let hora = 14;  
if (hora < 12) {  
 console.log("Bom dia!");  
} else if (hora < 18) {  
 console.log("Boa tarde!");  
} else {  
 console.log("Boa noite!");  
}  
// Saída: Boa tarde!  
  
let idade = 18;  
if (idade >= 18) {  
 console.log("Você é maior de idade.");  
} else {  
 console.log("Você é menor de idade.");  
}  
// Saída: Você é maior de idade.

**TypeScript:**

let horaTS: number = 14;  
if (horaTS < 12) {  
 console.log("Bom dia!");  
} else if (horaTS < 18) {  
 console.log("Boa tarde!");  
} else {  
 console.log("Boa noite!");  
}  
// Saída: Boa tarde!  
  
let idadeTS: number = 18;  
if (idadeTS >= 18) {  
 console.log("Você é maior de idade.");  
} else {  
 console.log("Você é menor de idade.");  
}  
// Saída: Você é maior de idade.

**PHP:**

$hora = 14;  
if ($hora < 12) {  
 echo "Bom dia!\n";  
} else if ($hora < 18) {  
 echo "Boa tarde!\n";  
} else {  
 echo "Boa noite!\n";  
}  
// Saída: Boa tarde!  
  
$idade = 18;  
if ($idade >= 18) {  
 echo "Você é maior de idade.\n";  
} else {  
 echo "Você é menor de idade.\n";  
}  
// Saída: Você é maior de idade.

**Observações:**

* As estruturas condicionais if, else if e else funcionam de forma idêntica nas três linguagens, com pequenas diferenças sintáticas (uso de $ para variáveis em PHP).

**Documentação Oficial:**

* JavaScript: [if…else](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/if...else)
* PHP: [if…else](https://www.php.net/manual/pt_BR/control-structures.if.php)

#### switch

Permite controlar o fluxo de execução com base em múltiplos valores possíveis de uma expressão.

**Sintaxe Geral:**

switch (expressao) {  
 case valor1:  
 // código a ser executado se expressao for igual a valor1  
 break;  
 case valor2:  
 // código a ser executado se expressao for igual a valor2  
 break;  
 default:  
 // código a ser executado se nenhum dos casos anteriores for correspondido  
}

**JavaScript/TypeScript:**

**JavaScript:**

let diaDaSemana = 3;  
let nomeDoDia;  
  
switch (diaDaSemana) {  
 case 1:  
 nomeDoDia = "Domingo";  
 break;  
 case 2:  
 nomeDoDia = "Segunda-feira";  
 break;  
 case 3:  
 nomeDoDia = "Terça-feira";  
 break;  
 case 4:  
 nomeDoDia = "Quarta-feira";  
 break;  
 case 5:  
 nomeDoDia = "Quinta-feira";  
 break;  
 case 6:  
 nomeDoDia = "Sexta-feira";  
 break;  
 case 7:  
 nomeDoDia = "Sábado";  
 break;  
 default:  
 nomeDoDia = "Dia inválido";  
}  
console.log(nomeDoDia); // Terça-feira

**TypeScript:**

let diaDaSemanaTS: number = 3;  
let nomeDoDiaTS: string;  
  
switch (diaDaSemanaTS) {  
 case 1:  
 nomeDoDiaTS = "Domingo";  
 break;  
 case 2:  
 nomeDoDiaTS = "Segunda-feira";  
 break;  
 case 3:  
 nomeDoDiaTS = "Terça-feira";  
 break;  
 case 4:  
 nomeDoDiaTS = "Quarta-feira";  
 break;  
 case 5:  
 nomeDoDiaTS = "Quinta-feira";  
 break;  
 case 6:  
 nomeDoDiaTS = "Sexta-feira";  
 break;  
 case 7:  
 nomeDoDiaTS = "Sábado";  
 break;  
 default:  
 nomeDoDiaTS = "Dia inválido";  
}  
console.log(nomeDoDiaTS); // Terça-feira

**PHP:**

$diaDaSemana = 3;  
$nomeDoDia;  
  
switch ($diaDaSemana) {  
 case 1:  
 $nomeDoDia = "Domingo";  
 break;  
 case 2:  
 $nomeDoDia = "Segunda-feira";  
 break;  
 case 3:  
 $nomeDoDia = "Terça-feira";  
 break;  
 case 4:  
 $nomeDoDia = "Quarta-feira";  
 break;  
 case 5:  
 $nomeDoDia = "Quinta-feira";  
 break;  
 case 6:  
 $nomeDoDia = "Sexta-feira";  
 break;  
 case 7:  
 $nomeDoDia = "Sábado";  
 break;  
 default:  
 $nomeDoDia = "Dia inválido";  
}  
echo $nomeDoDia . "\n"; // Terça-feira

**Observações:**

* A estrutura switch funciona de forma idêntica nas três linguagens. É importante usar break para evitar a execução de blocos de código subsequentes (fall-through).

**Documentação Oficial:**

* JavaScript: [switch](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/switch)
* PHP: [switch](https://www.php.net/manual/pt_BR/control-structures.switch.php)

### 3.2. Estruturas de Repetição

#### for

Executa um bloco de código um número específico de vezes.

**Sintaxe Geral:**

for (inicializacao; condicao; incremento) {  
 // código a ser executado  
}

**JavaScript/TypeScript:**

**JavaScript:**

for (let i = 0; i < 5; i++) {  
 console.log("JavaScript: " + i);  
}

**TypeScript:**

for (let i: number = 0; i < 5; i++) {  
 console.log("TypeScript: " + i);  
}

**PHP:**

for ($i = 0; $i < 5; $i++) {  
 echo "PHP: " . $i . "\n";  
}

**Observações:**

* O loop for funciona de forma idêntica nas três linguagens.

**Documentação Oficial:**

* JavaScript: [for](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/for)
* PHP: [for](https://www.php.net/manual/pt_BR/control-structures.for.php)

#### while

Executa um bloco de código enquanto uma condição especificada for verdadeira.

**Sintaxe Geral:**

while (condicao) {  
 // código a ser executado  
}

**JavaScript/TypeScript:**

let i = 0;  
while (i < 5) {  
 console.log("JavaScript: " + i);  
 i++;  
}

**TypeScript:**

let iTS: number = 0;  
while (iTS < 5) {  
 console.log("TypeScript: " + iTS);  
 iTS++;  
}

**PHP:**

$i = 0;  
while ($i < 5) {  
 echo "PHP: " . $i . "\n";  
 $i++;  
}

**Observações:**

* O loop while funciona de forma idêntica nas três linguagens.

**Documentação Oficial:**

* JavaScript: [while](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/while)
* PHP: [while](https://www.php.net/manual/pt_BR/control-structures.while.php)

#### do...while

Executa um bloco de código uma vez, e então repete o loop enquanto a condição especificada for verdadeira. Garante que o bloco de código seja executado pelo menos uma vez.

**Sintaxe Geral:**

do {  
 // código a ser executado  
} while (condicao);

**JavaScript/TypeScript:**

let i = 0;  
do {  
 console.log("JavaScript: " + i);  
 i++;  
} while (i < 5);

**TypeScript:**

let iTS: number = 0;  
do {  
 console.log("TypeScript: " + iTS);  
 iTS++;  
} while (iTS < 5);

**PHP:**

$i = 0;  
do {  
 echo "PHP: " . $i . "\n";  
 $i++;  
} while ($i < 5);

**Observações:**

* O loop do...while funciona de forma idêntica nas três linguagens.

**Documentação Oficial:**

* JavaScript: [do…while](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/do...while)
* PHP: [do…while](https://www.php.net/manual/pt_BR/control-structures.do.while.php)

#### for...of (JavaScript/TypeScript) e foreach (PHP)

Iteram sobre coleções de dados.

**JavaScript/TypeScript (for...of):**

const arrayJS = [1, 2, 3];  
for (const elemento of arrayJS) {  
 console.log("JavaScript (for...of): " + elemento);  
}  
  
const stringJS = "hello";  
for (const char of stringJS) {  
 console.log("JavaScript (for...of string): " + char);  
}

**TypeScript:**

const arrayTS: number[] = [1, 2, 3];  
for (const elemento of arrayTS) {  
 console.log("TypeScript (for...of): " + elemento);  
}  
  
const stringTS: string = "hello";  
for (const char of stringTS) {  
 console.log("TypeScript (for...of string): " + char);  
}

**PHP (foreach):**

$arrayPHP = [1, 2, 3];  
foreach ($arrayPHP as $elemento) {  
 echo "PHP (foreach): " . $elemento . "\n";  
}  
  
$associativeArrayPHP = [  
 "a" => 1,  
 "b" => 2,  
 "c" => 3  
];  
foreach ($associativeArrayPHP as $chave => $valor) {  
 echo "PHP (foreach associativo): " . $chave . " => " . $valor . "\n";  
}

**Observações:**

* for...of em JavaScript/TypeScript é usado para iterar sobre valores de objetos iteráveis (Arrays, Strings, Maps, Sets, etc.).
* foreach em PHP é usado para iterar sobre elementos de arrays e objetos. Pode ser usado para acessar apenas valores ou pares chave-valor.

**Documentação Oficial:**

* JavaScript: [for…of](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/for...of)
* PHP: [foreach](https://www.php.net/manual/pt_BR/control-structures.foreach.php)

#### for...in (JavaScript/TypeScript)

Itera sobre as propriedades enumeráveis de um objeto.

**JavaScript/TypeScript:**

const objetoJS = { a: 1, b: 2, c: 3 };  
for (const chave in objetoJS) {  
 console.log(`JavaScript (for...in): ${chave}: ${objetoJS[chave]}`);  
}

**TypeScript:**

const objetoTS: { [key: string]: number } = { a: 1, b: 2, c: 3 };  
for (const chave in objetoTS) {  
 console.log(`TypeScript (for...in): ${chave}: ${objetoTS[chave]}`);  
}

**Observações:**

* for...in em JavaScript/TypeScript é usado para iterar sobre as chaves (nomes das propriedades) de um objeto. Não é recomendado para iterar sobre arrays, pois pode incluir propriedades herdadas e a ordem não é garantida.

**Documentação Oficial:**

* JavaScript: [for…in](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/for...in)

### 3.3. Controle de Fluxo

#### break

Termina a execução do loop atual ou da instrução switch e transfere o controle para a instrução que segue o loop ou switch.

**JavaScript/TypeScript:**

for (let i = 0; i < 10; i++) {  
 if (i === 5) {  
 break; // Sai do loop quando i for 5  
 }  
 console.log("JavaScript (break): " + i);  
}  
// Saída: 0, 1, 2, 3, 4

**TypeScript:**

for (let i: number = 0; i < 10; i++) {  
 if (i === 5) {  
 break; // Sai do loop quando i for 5  
 }  
 console.log("TypeScript (break): " + i);  
}  
// Saída: 0, 1, 2, 3, 4

**PHP:**

for ($i = 0; $i < 10; $i++) {  
 if ($i === 5) {  
 break; // Sai do loop quando $i for 5  
 }  
 echo "PHP (break): " . $i . "\n";  
}  
// Saída: 0, 1, 2, 3, 4

**Observações:**

* O break funciona de forma idêntica nas três linguagens.

**Documentação Oficial:**

* JavaScript: [break](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/break)
* PHP: [break](https://www.php.net/manual/pt_BR/control-structures.break.php)

#### continue

Termina a execução da iteração atual do loop e continua a execução do loop na próxima iteração.

**JavaScript/TypeScript:**

for (let i = 0; i < 5; i++) {  
 if (i === 2) {  
 continue; // Pula a iteração quando i for 2  
 }  
 console.log("JavaScript (continue): " + i);  
}  
// Saída: 0, 1, 3, 4

**TypeScript:**

for (let i: number = 0; i < 5; i++) {  
 if (i === 2) {  
 continue; // Pula a iteração quando i for 2  
 }  
 console.log("TypeScript (continue): " + i);  
}  
// Saída: 0, 1, 3, 4

**PHP:**

for ($i = 0; $i < 5; $i++) {  
 if ($i === 2) {  
 continue; // Pula a iteração quando $i for 2  
 }  
 echo "PHP (continue): " . $i . "\n";  
}  
// Saída: 0, 1, 3, 4

**Observações:**

* O continue funciona de forma idêntica nas três linguagens.

**Documentação Oficial:**

* JavaScript: [continue](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/continue)
* PHP: [continue](https://www.php.net/manual/pt_BR/control-structures.continue.php)

## 4. Programação Orientada a Objetos (POO)

### Conceitos Fundamentais da POO (Classes, Objetos, Atributos, Métodos)

#### Classes

Uma classe é um molde ou um projeto para criar objetos. Ela define as propriedades (atributos) e os comportamentos (métodos) que os objetos criados a partir dela terão.

**JavaScript/TypeScript:**

class Pessoa {  
 constructor(nome, idade) {  
 this.nome = nome;  
 this.idade = idade;  
 }  
  
 apresentar() {  
 console.log(`Olá, meu nome é ${this.nome} e tenho ${this.idade} anos.`);  
 }  
}

**TypeScript:**

class PessoaTS {  
 nome: string;  
 idade: number;  
  
 constructor(nome: string, idade: number) {  
 this.nome = nome;  
 this.idade = idade;  
 }  
  
 apresentar(): void {  
 console.log(`Olá, meu nome é ${this.nome} e tenho ${this.idade} anos.`);  
 }  
}

**PHP:**

class Pessoa {  
 public $nome;  
 public $idade;  
  
 public function \_\_construct($nome, $idade) {  
 $this->nome = $nome;  
 $this->idade = $idade;  
 }  
  
 public function apresentar() {  
 echo "Olá, meu nome é " . $this->nome . " e tenho " . $this->idade . " anos.\n";  
 }  
}

**Observações:**

* Tanto JavaScript/TypeScript quanto PHP utilizam a palavra-chave class para definir classes. Em PHP, é necessário declarar a visibilidade (public, private, protected) dos atributos e métodos.

#### Objetos

Um objeto é uma instância de uma classe. É uma entidade concreta criada a partir do molde definido pela classe.

**JavaScript/TypeScript:**

const pessoa1 = new Pessoa("Alice", 30);  
pessoa1.apresentar(); // Olá, meu nome é Alice e tenho 30 anos.

**TypeScript:**

const pessoa1TS: PessoaTS = new PessoaTS("Alice", 30);  
pessoa1TS.apresentar(); // Olá, meu nome é Alice e tenho 30 anos.

**PHP:**

$pessoa1 = new Pessoa("Alice", 30);  
$pessoa1->apresentar(); // Olá, meu nome é Alice e tenho 30 anos.

**Observações:**

* A criação de objetos é similar em ambas as linguagens, utilizando a palavra-chave new.

#### Atributos (Propriedades)

São as características ou dados que um objeto possui. Representam o estado do objeto.

**JavaScript/TypeScript:**

// Exemplo na classe Pessoa: this.nome e this.idade são atributos  
console.log(pessoa1.nome); // Alice  
console.log(pessoa1.idade); // 30

**TypeScript:**

// Exemplo na classe PessoaTS: this.nome e this.idade são atributos  
console.log(pessoa1TS.nome); // Alice  
console.log(pessoa1TS.idade); // 30

**PHP:**

// Exemplo na classe Pessoa: $this->nome e $this->idade são atributos  
echo $pessoa1->nome . "\n"; // Alice  
echo $pessoa1->idade . "\n"; // 30

**Observações:**

* O acesso aos atributos é feito de forma similar, usando a notação de ponto (.) em JavaScript/TypeScript e a notação de seta (->) em PHP.

#### Métodos

São as ações ou comportamentos que um objeto pode realizar. Representam a funcionalidade do objeto.

**JavaScript/TypeScript:**

// Exemplo na classe Pessoa: apresentar() é um método  
pessoa1.apresentar(); // Olá, meu nome é Alice e tenho 30 anos.

**TypeScript:**

// Exemplo na classe PessoaTS: apresentar() é um método  
pessoa1TS.apresentar(); // Olá, meu nome é Alice e tenho 30 anos.

**PHP:**

// Exemplo na classe Pessoa: apresentar() é um método  
$pessoa1->apresentar(); // Olá, meu nome é Alice e tenho 30 anos.

**Observações:**

* A chamada de métodos é similar em ambas as linguagens, usando a notação de ponto (.) em JavaScript/TypeScript e a notação de seta (->) em PHP.

### Encapsulamento

Encapsulamento é o princípio de agrupar dados (atributos) e os métodos que operam nesses dados em uma única unidade (a classe), e restringir o acesso direto a alguns dos componentes do objeto. Isso protege a integridade dos dados e permite que a implementação interna da classe seja alterada sem afetar o código externo que a utiliza.

**JavaScript/TypeScript:**

Em JavaScript, o encapsulamento tradicional (com modificadores de acesso public, private, protected) não existe nativamente da mesma forma que em linguagens como Java ou PHP. No entanto, existem convenções e recursos que permitem simular o encapsulamento:

* **Convenção:** Usar um prefixo \_ (underscore) para indicar que um atributo ou método é

privado ou protegido. Isso é apenas uma convenção e não impede o acesso direto.

* **Closures:** Utilizar closures para criar variáveis e funções privadas.
* **Campos de Classe Privados (ES2019+):** Com a introdução de campos de classe privados (#), JavaScript agora oferece um mecanismo nativo para encapsulamento real.

class ContaBancaria {  
 #saldo; // Campo de classe privado  
  
 constructor(saldoInicial) {  
 this.#saldo = saldoInicial;  
 }  
  
 depositar(valor) {  
 if (valor > 0) {  
 this.#saldo += valor;  
 console.log(`Depósito de ${valor} realizado. Novo saldo: ${this.#saldo}`);  
 } else {  
 console.log("Valor de depósito inválido.");  
 }  
 }  
  
 sacar(valor) {  
 if (valor > 0 && valor <= this.#saldo) {  
 this.#saldo -= valor;  
 console.log(`Saque de ${valor} realizado. Novo saldo: ${this.#saldo}`);  
 } else {  
 console.log("Saldo insuficiente ou valor de saque inválido.");  
 }  
 }  
  
 getSaldo() {  
 return this.#saldo;  
 }  
}  
  
const minhaConta = new ContaBancaria(1000);  
minhaConta.depositar(200);  
minhaConta.sacar(500);  
// console.log(minhaConta.#saldo); // Erro: Campo privado não pode ser acessado diretamente  
console.log(`Saldo atual: ${minhaConta.getSaldo()}`);

**TypeScript:**

class ContaBancariaTS {  
 private \_saldo: number; // Modificador private em TypeScript  
  
 constructor(saldoInicial: number) {  
 this.\_saldo = saldoInicial;  
 }  
  
 depositar(valor: number): void {  
 if (valor > 0) {  
 this.\_saldo += valor;  
 console.log(`Depósito de ${valor} realizado. Novo saldo: ${this.\_saldo}`);  
 } else {  
 console.log("Valor de depósito inválido.");  
 }  
 }  
  
 sacar(valor: number): void {  
 if (valor > 0 && valor <= this.\_saldo) {  
 this.\_saldo -= valor;  
 console.log(`Saque de ${valor} realizado. Novo saldo: ${this.\_saldo}`);  
 } else {  
 console.log("Saldo insuficiente ou valor de saque inválido.");  
 }  
 }  
  
 getSaldo(): number {  
 return this.\_saldo;  
 }  
}  
  
const minhaContaTS = new ContaBancariaTS(1000);  
minhaContaTS.depositar(200);  
minhaContaTS.sacar(500);  
// console.log(minhaContaTS.\_saldo); // Erro em tempo de compilação: Propriedade privada  
console.log(`Saldo atual: ${minhaContaTS.getSaldo()}`);

**TypeScript:**

TypeScript, além dos campos de classe privados do JavaScript, oferece modificadores de acesso (public, private, protected) que são aplicados em tempo de compilação para garantir o encapsulamento. No entanto, após a compilação para JavaScript, esses modificadores não existem mais, e o encapsulamento real depende dos recursos do JavaScript (como os campos privados).

class ContaBancariaTS {  
 private \_saldo: number; // Modificador private em TypeScript  
  
 constructor(saldoInicial: number) {  
 this.\_saldo = saldoInicial;  
 }  
  
 depositar(valor: number): void {  
 if (valor > 0) {  
 this.\_saldo += valor;  
 console.log(`Depósito de ${valor} realizado. Novo saldo: ${this.\_saldo}`);  
 } else {  
 console.log("Valor de depósito inválido.");  
 }  
 }  
  
 sacar(valor: number): void {  
 if (valor > 0 && valor <= this.\_saldo) {  
 this.\_saldo -= valor;  
 console.log(`Saque de ${valor} realizado. Novo saldo: ${this.\_saldo}`);  
 } else {  
 console.log("Saldo insuficiente ou valor de saque inválido.");  
 }  
 }  
  
 getSaldo(): number {  
 return this.\_saldo;  
 }  
}  
  
const minhaContaTS = new ContaBancariaTS(1000);  
minhaContaTS.depositar(200);  
minhaContaTS.sacar(500);  
// console.log(minhaContaTS.\_saldo); // Erro em tempo de compilação: Propriedade privada  
console.log(`Saldo atual: ${minhaContaTS.getSaldo()}`);

**PHP:**

PHP possui modificadores de acesso explícitos (public, private, protected) que controlam a visibilidade de propriedades e métodos.

* public: A propriedade ou método pode ser acessado de qualquer lugar.
* private: A propriedade ou método só pode ser acessado de dentro da própria classe.
* protected: A propriedade ou método pode ser acessado de dentro da própria classe e de classes que herdam dela.

class ContaBancariaPHP {  
 private $saldo; // Propriedade privada  
  
 public function \_\_construct($saldoInicial) {  
 $this->saldo = $saldoInicial;  
 }  
  
 public function depositar($valor) {  
 if ($valor > 0) {  
 $this->saldo += $valor;  
 echo "Depósito de " . $valor . " realizado. Novo saldo: " . $this->saldo . "\n";  
 } else {  
 echo "Valor de depósito inválido.\n";  
 }  
 }  
  
 public function sacar($valor) {  
 if ($valor > 0 && $valor <= $this->saldo) {  
 $this->saldo -= $valor;  
 echo "Saque de " . $valor . " realizado. Novo saldo: " . $this->saldo . "\n";  
 } else {  
 echo "Saldo insuficiente ou valor de saque inválido.\n";  
 }  
 }  
  
 public function getSaldo() {  
 return $this->saldo;  
 }  
}  
  
$minhaContaPHP = new ContaBancariaPHP(1000);  
$minhaContaPHP->depositar(200);  
$minhaContaPHP->sacar(500);  
// echo $minhaContaPHP->saldo; // Erro fatal: Não é possível acessar propriedade privada  
echo "Saldo atual: " . $minhaContaPHP->getSaldo() . "\n";

**Observações:**

* O encapsulamento é fundamental para a segurança e manutenção do código, garantindo que o estado interno de um objeto seja manipulado apenas por seus próprios métodos.

**Documentação Oficial:**

* JavaScript: [Campos de Classe Privados](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Classes/Private_class_fields)
* TypeScript: [Modificadores de Acesso](https://www.typescriptlang.org/docs/handbook/classes.html#member-visibility)
* PHP: [Visibilidade](https://www.php.net/manual/pt_BR/language.oop5.visibility.php)

### Herança

Herança é um mecanismo que permite que uma classe (subclasse ou classe filha) herde propriedades e métodos de outra classe (superclasse ou classe pai). Isso promove a reutilização de código e estabelece uma relação “é um tipo de” entre as classes.

**JavaScript/TypeScript:**

Utilizam a palavra-chave extends para herança.

class Animal {  
 constructor(nome) {  
 this.nome = nome;  
 }  
  
 fazerBarulho() {  
 console.log("Algum barulho...");  
 }  
}  
  
class Cachorro extends Animal {  
 constructor(nome, raca) {  
 super(nome);  
 this.raca = raca;  
 }  
  
 fazerBarulho() {  
 console.log("Au au!");  
 }  
  
 latir() {  
 console.log("O cachorro " + this.nome + " está latindo!");  
 }  
}  
  
const meuCachorro = new Cachorro("Rex", "Labrador");  
meuCachorro.fazerBarulho(); // Au au!  
meuCachorro.latir(); // O cachorro Rex está latindo!  
console.log(meuCachorro.nome); // Rex

**TypeScript:**

class AnimalTSHeranca {  
 constructor(protected nome: string) {}  
  
 fazerBarulho(): void {  
 console.log("Algum barulho...");  
 }  
}  
  
class CachorroTSHeranca extends AnimalTSHeranca {  
 constructor(nome: string, public raca: string) {  
 super(nome); // Chama o construtor da classe pai  
 }  
  
 fazerBarulho(): void {  
 console.log("Au au!");  
 }  
  
 latir(): void {  
 console.log(`O cachorro ${this.nome} está latindo!`);  
 }  
}  
  
const meuCachorroTSHeranca: CachorroTSHeranca = new CachorroTSHeranca("Rex", "Labrador");  
meuCachorroTSHeranca.fazerBarulho(); // Au au!  
meuCachorroTSHeranca.latir(); // O cachorro Rex está latindo!  
console.log(meuCachorroTSHeranca.nome); // Rex

**PHP:**

Também utiliza a palavra-chave extends para herança.

class Animal {  
 public $nome;  
  
 public function \_\_construct($nome) {  
 $this->nome = $nome;  
 }  
  
 public function fazerBarulho() {  
 echo "Algum barulho...\n";  
 }  
}  
  
class Cachorro extends Animal {  
 public $raca;  
  
 public function \_\_construct($nome, $raca) {  
 parent::\_\_construct($nome); // Chama o construtor da classe pai  
 $this->raca = $raca;  
 }  
  
 public function fazerBarulho() {  
 echo "Au au!\n";  
 }  
  
 public function latir() {  
 echo "O cachorro " . $this->nome . " está latindo!\n";  
 }  
}  
  
$meuCachorro = new Cachorro("Rex", "Labrador");  
$meuCachorro->fazerBarulho(); // Au au!  
$meuCachorro->latir(); // O cachorro Rex está latindo!  
echo $meuCachorro->nome . "\n"; // Rex

**Observações:**

* Ambas as linguagens usam extends para herdar. Em JavaScript/TypeScript, super() é usado para chamar o construtor da classe pai. Em PHP, parent::\_\_construct() é usado para o mesmo propósito.
* Métodos podem ser sobrescritos nas subclasses para fornecer implementações específicas.

**Documentação Oficial:**

* JavaScript: [extends](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Classes/extends)
* PHP: [Herança de Objetos](https://www.php.net/manual/pt_BR/language.oop5.inheritance.php)

### Polimorfismo

Polimorfismo significa “muitas formas”. Em POO, refere-se à capacidade de objetos de diferentes classes responderem ao mesmo método de maneiras diferentes, desde que essas classes compartilhem uma interface comum ou herdem de uma superclasse comum. Isso permite que o código seja mais flexível e extensível.

**JavaScript/TypeScript:**

O polimorfismo é naturalmente suportado através da herança e da sobrescrita de métodos.

class AnimalGenerico {  
 fazerBarulho() {  
 console.log("Barulho genérico de animal.");  
 }  
}  
  
class Gato extends AnimalGenerico {  
 fazerBarulho() {  
 console.log("Miau!");  
 }  
}  
  
class Pato extends AnimalGenerico {  
 fazerBarulho() {  
 console.log("Quack!");  
 }  
}  
  
function emitirBarulho(animal) {  
 animal.fazerBarulho();  
}  
  
const meuGato = new Gato();  
const meuPato = new Pato();  
const meuAnimalGenerico = new AnimalGenerico();  
  
emitirBarulho(meuGato); // Miau!  
emitirBarulho(meuPato); // Quack!  
emitirBarulho(meuAnimalGenerico); // Barulho genérico de animal.

**TypeScript:**

class AnimalGenericoTS {  
 fazerBarulho(): void {  
 console.log("Barulho genérico de animal.");  
 }  
}  
  
class GatoTS extends AnimalGenericoTS {  
 fazerBarulho(): void {  
 console.log("Miau!");  
 }  
}  
  
class PatoTS extends AnimalGenericoTS {  
 fazerBarulho(): void {  
 console.log("Quack!");  
 }  
}  
  
function emitirBarulhoTS(animal: AnimalGenericoTS) {  
 animal.fazerBarulho();  
}  
  
const meuGatoTS: GatoTS = new GatoTS();  
const meuPatoTS: PatoTS = new PatoTS();  
const meuAnimalGenericoTS: AnimalGenericoTS = new AnimalGenericoTS();  
  
emitirBarulhoTS(meuGatoTS); // Miau!  
emitirBarulhoTS(meuPatoTS); // Quack!  
emitirBarulhoTS(meuAnimalGenericoTS); // Barulho genérico de animal.

**PHP:**

O polimorfismo também é suportado através da herança e da sobrescrita de métodos, e pode ser reforçado com interfaces.

class AnimalGenericoPHP {  
 public function fazerBarulho() {  
 echo "Barulho genérico de animal.\n";  
 }  
}  
  
class GatoPHP extends AnimalGenericoPHP {  
 public function fazerBarulho() {  
 echo "Miau!\n";  
 }  
}  
  
class PatoPHP extends AnimalGenericoPHP {  
 public function fazerBarulho() {  
 echo "Quack!\n";  
 }  
}  
  
function emitirBarulhoPHP(AnimalGenericoPHP $animal) {  
 $animal->fazerBarulho();  
}  
  
$meuGatoPHP = new GatoPHP();  
$meuPatoPHP = new PatoPHP();  
$meuAnimalGenericoPHP = new AnimalGenericoPHP();  
  
emitirBarulhoPHP($meuGatoPHP); // Miau!  
emitirBarulhoPHP($meuPatoPHP); // Quack!  
emitirBarulhoPHP($meuAnimalGenericoPHP); // Barulho genérico de animal.

**Observações:**

* O polimorfismo permite que você escreva código mais genérico que pode operar em objetos de diferentes tipos, desde que eles compartilhem uma base comum (herança ou interface).

**Documentação Oficial:**

* JavaScript: [Polimorfismo (conceito)](https://developer.mozilla.org/pt-BR/docs/Glossary/Polymorphism)
* PHP: [Polimorfismo (conceito)](https://www.php.net/manual/pt_BR/language.oop5.interfaces.php#language.oop5.interfaces.polymorphism)

### Interfaces e Classes Abstratas (onde aplicável)

#### Interfaces

Interfaces definem um contrato que as classes devem seguir. Elas especificam quais métodos uma classe deve implementar, mas não fornecem a implementação desses métodos. Uma classe pode implementar várias interfaces.

**TypeScript:**

TypeScript possui o conceito de interfaces para definir a forma de objetos e classes. Em tempo de execução, as interfaces são removidas, pois JavaScript não tem interfaces nativas.

interface FormaGeometrica {  
 calcularArea(): number;  
 calcularPerimetro(): number;  
}  
  
class Circulo implements FormaGeometrica {  
 constructor(private raio: number) {}  
  
 calcularArea(): number {  
 return Math.PI \* this.raio \* this.raio;  
 }  
  
 calcularPerimetro(): number {  
 return 2 \* Math.PI \* this.raio;  
 }  
}  
  
class Retangulo implements FormaGeometrica {  
 constructor(private largura: number, private altura: number) {}  
  
 calcularArea(): number {  
 return this.largura \* this.altura;  
 }  
  
 calcularPerimetro(): number {  
 return 2 \* (this.largura + this.altura);  
 }  
}  
  
const meuCirculo = new Circulo(5);  
console.log(`Área do Círculo: ${meuCirculo.calcularArea()}`);  
console.log(`Perímetro do Círculo: ${meuCirculo.calcularPerimetro()}`);  
  
const meuRetangulo = new Retangulo(4, 6);  
console.log(`Área do Retângulo: ${meuRetangulo.calcularArea()}`);  
console.log(`Perímetro do Retângulo: ${meuRetangulo.calcularPerimetro()}`);

**PHP:**

PHP possui interfaces nativas que são usadas para definir contratos para classes.

interface FormaGeometricaPHP {  
 public function calcularArea(): float;  
 public function calcularPerimetro(): float;  
}  
  
class CirculoPHP implements FormaGeometricaPHP {  
 private $raio;  
  
 public function \_\_construct(float $raio) {  
 $this->raio = $raio;  
 }  
  
 public function calcularArea(): float {  
 return M\_PI \* $this->raio \* $this->raio;  
 }  
  
 public function calcularPerimetro(): float {  
 return 2 \* M\_PI \* $this->raio;  
 }  
}  
  
class RetanguloPHP implements FormaGeometricaPHP {  
 private $largura;  
 private $altura;  
  
 public function \_\_construct(float $largura, float $altura) {  
 $this->largura = $largura;  
 $this->altura = $altura;  
 }  
  
 public function calcularArea(): float {  
 return $this->largura \* $this->altura;  
 }  
  
 public function calcularPerimetro(): float {  
 return 2 \* ($this->largura + $this->altura);  
 }  
}  
  
$meuCirculoPHP = new CirculoPHP(5);  
echo "Área do Círculo: " . $meuCirculoPHP->calcularArea() . "\n";  
echo "Perímetro do Círculo: " . $meuCirculoPHP->calcularPerimetro() . "\n";  
  
$meuRetanguloPHP = new RetanguloPHP(4, 6);  
echo "Área do Retângulo: " . $meuRetanguloPHP->calcularArea() . "\n";  
echo "Perímetro do Retângulo: " . $meuRetanguloPHP->calcularPerimetro() . "\n";

**Observações:**

* Interfaces são úteis para definir contratos e garantir que as classes que as implementam forneçam métodos específicos.

**Documentação Oficial:**

* TypeScript: [Interfaces](https://www.typescriptlang.org/docs/handbook/interfaces.html)
* PHP: [Interfaces de Objeto](https://www.php.net/manual/pt_BR/language.oop5.interfaces.php)

#### Classes Abstratas

Classes abstratas são classes que não podem ser instanciadas diretamente e podem conter métodos abstratos (métodos sem implementação) e métodos concretos (métodos com implementação). Elas servem como base para outras classes, que devem implementar os métodos abstratos.

**TypeScript:**

TypeScript suporta classes abstratas.

abstract class AnimalAbstrato {  
 constructor(protected nome: string) {}  
  
 abstract fazerBarulho(): void; // Método abstrato  
  
 mover(): void {  
 console.log(`${this.nome} está se movendo.`);  
 }  
}  
  
class CachorroConcreto extends AnimalAbstrato {  
 constructor(nome: string, private raca: string) {  
 super(nome);  
 }  
  
 fazerBarulho(): void {  
 console.log("Au au!");  
 }  
  
 mostrarRaca(): void {  
 console.log(`A raça do ${this.nome} é ${this.raca}.`);  
 }  
}  
  
// const animal = new AnimalAbstrato("Bicho"); // Erro: Não é possível instanciar uma classe abstrata  
  
const meuCachorroConcreto = new CachorroConcreto("Buddy", "Golden Retriever");  
meuCachorroConcreto.fazerBarulho(); // Au au!  
meuCachorroConcreto.mover(); // Buddy está se movendo.  
meuCachorroConcreto.mostrarRaca(); // A raça do Buddy é Golden Retriever.

**PHP:**

PHP também suporta classes abstratas.

abstract class AnimalAbstratoPHP {  
 protected $nome;  
  
 public function \_\_construct($nome) {  
 $this->nome = $nome;  
 }  
  
 abstract public function fazerBarulho(); // Método abstrato  
  
 public function mover() {  
 echo $this->nome . " está se movendo.\n";  
 }  
}  
  
class CachorroConcretoPHP extends AnimalAbstratoPHP {  
 private $raca;  
  
 public function \_\_construct($nome, $raca) {  
 parent::\_\_construct($nome);  
 $this->raca = $raca;  
 }  
  
 public function fazerBarulho() {  
 echo "Au au!\n";  
 }  
  
 public function mostrarRaca() {  
 echo "A raça do " . $this->nome . " é " . $this->raca . ".\n";  
 }  
}  
  
// $animalPHP = new AnimalAbstratoPHP("Bicho"); // Erro fatal: Não é possível instanciar uma classe abstrata  
  
$meuCachorroConcretoPHP = new CachorroConcretoPHP("Buddy", "Golden Retriever");  
$meuCachorroConcretoPHP->fazerBarulho(); // Au au!  
$meuCachorroConcretoPHP->mover(); // Buddy está se movendo.  
$meuCachorroConcretoPHP->mostrarRaca(); // A raça do Buddy é Golden Retriever.

**Observações:**

* Classes abstratas são usadas para definir uma estrutura comum para um conjunto de classes relacionadas, forçando as subclasses a implementar certos métodos.

**Documentação Oficial:**

* TypeScript: [Classes Abstratas](https://www.typescriptlang.org/docs/handbook/classes.html#abstract-classes)
* PHP: [Classes Abstratas](https://www.php.net/manual/pt_BR/language.oop5.abstract.php)

## 5. Referencias Usadas

**Autor:** Manus AI

**Referências:**

[1] [MDN Web Docs - JavaScript](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript) [2] [TypeScript Documentation](https://www.typescriptlang.org/docs/) [3] [Manual do PHP](https://www.php.net/manual/pt_BR/) [4] [JavaScript: Operadores Aritméticos](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/Arithmetic_operators) [5] [PHP: Operadores Aritméticos](https://www.php.net/manual/pt_BR/language.operators.arithmetic.php) [6] [JavaScript: Operadores de Atribuição](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/Assignment_operators) [7] [PHP: Operadores de Atribuição](https://www.php.net/manual/pt_BR/language.operators.assignment.php) [8] [JavaScript: Operadores de Comparação](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/Comparison_operators) [9] [PHP: Operadores de Comparação](https://www.php.net/manual/pt_BR/language.operators.comparison.php) [10] [JavaScript: Operadores Lógicos](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/Logical_AND) [11] [PHP: Operadores Lógicos](https://www.php.net/manual/pt_BR/language.operators.logical.php) [12] [JavaScript: Operador de Concatenação](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/Addition) [13] [PHP: Operadores de String](https://www.php.net/manual/pt_BR/language.operators.string.php) [14] [JavaScript: Operadores de Incremento e Decremento](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/Increment) [15] [PHP: Operadores de Incremento/Decremento](https://www.php.net/manual/pt_BR/language.operators.increment.php) [16] [JavaScript: Operador Condicional (Ternário)](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/Conditional_operator) [17] [PHP: Operador Ternário](https://www.php.net/manual/pt_BR/language.operators.comparison.php#language.operators.comparison.ternary) [18] [JavaScript: Operador typeof](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/typeof) [19] [JavaScript: Operador instanceof](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/instanceof) [20] [PHP: Funções de Tipo](https://www.php.net/manual/pt_BR/ref.var.php) [21] [PHP: Operador instanceof](https://www.php.net/manual/pt_BR/language.operators.type.php) [22] [JavaScript: if…else](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/if...else) [23] [PHP: if…else](https://www.php.net/manual/pt_BR/control-structures.if.php) [24] [JavaScript: switch](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/switch) [25] [PHP: switch](https://www.php.net/manual/pt_BR/control-structures.switch.php) [26] [JavaScript: for](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/for) [27] [PHP: for](https://www.php.net/manual/pt_BR/control-structures.for.php) [28] [JavaScript: while](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/while) [29] [PHP: while](https://www.php.net/manual/pt_BR/control-structures.while.php) [30] [JavaScript: do…while](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/do...while) [31] [PHP: do…while](https://www.php.net/manual/pt_BR/control-structures.do.while.php) [32] [JavaScript: for…of](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/for...of) [33] [PHP: foreach](https://www.php.net/manual/pt_BR/control-structures.foreach.php) [34] [JavaScript: for…in](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/for...in) [35] [JavaScript: break](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/break) [36] [PHP: break](https://www.php.net/manual/pt_BR/control-structures.break.php) [37] [JavaScript: continue](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/continue) [38] [PHP: continue](https://www.php.net/manual/pt_BR/control-structures.continue.php) [39] [JavaScript: Campos de Classe Privados](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Classes/Private_class_fields) [40] [TypeScript: Modificadores de Acesso](https://www.typescriptlang.org/docs/handbook/classes.html#member-visibility) [41] [PHP: Visibilidade](https://www.php.net/manual/pt_BR/language.oop5.visibility.php) [42] [JavaScript: extends](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Classes/extends) [43] [PHP: Herança de Objetos](https://www.php.net/manual/pt_BR/language.oop5.inheritance.php) [44] [JavaScript: Polimorfismo (conceito)](https://developer.mozilla.org/pt-BR/docs/Glossary/Polymorphism) [45] [PHP: Polimorfismo (conceito)](https://www.php.net/manual/pt_BR/language.oop5.interfaces.php#language.oop5.interfaces.polymorphism) [46] [TypeScript: Interfaces](https://www.typescriptlang.org/docs/handbook/interfaces.html) [47] [PHP: Interfaces de Objeto](https://www.php.net/manual/pt_BR/language.oop5.interfaces.php) [48] [TypeScript: Classes Abstratas](https://www.typescriptlang.org/docs/handbook/classes.html#abstract-classes) [49] [PHP: Classes Abstratas](https://www.php.net/manual/pt_BR/language.oop5.abstract.php)

### Orientação a Objetos por Prototipação (JavaScript)

Ao contrário de linguagens como Java, C++ ou PHP, que são baseadas em classes, o JavaScript é uma linguagem baseada em protótipos. Isso significa que, em vez de classes, o JavaScript utiliza objetos existentes (protótipos) como modelos para criar novos objetos. A herança é alcançada através da cadeia de protótipos, onde um objeto pode herdar propriedades e métodos de outro objeto.

Mesmo com a introdução da sintaxe class no ES2015 (ES6), é importante entender que essa é apenas uma “açúcar sintático” (syntactic sugar) sobre o modelo de protótipos existente. Por baixo dos panos, o JavaScript ainda utiliza protótipos para implementar classes e herança.

#### Como funciona a Prototipação

Cada objeto em JavaScript tem uma propriedade interna [[Prototype]] (acessível via \_\_proto\_\_ em navegadores, ou Object.getPrototypeOf()), que aponta para outro objeto, seu protótipo. Quando você tenta acessar uma propriedade ou método em um objeto, e ele não é encontrado diretamente nesse objeto, o JavaScript procura na cadeia de protótipos até encontrar a propriedade ou método ou chegar ao final da cadeia (null).

#### Exemplo de Herança Prototípica (antes do ES6 class)

// Objeto protótipo para Animais  
const AnimalProto = {  
 fazerBarulho: function() {  
 console.log("Barulho genérico de animal.");  
 },  
 comer: function() {  
 console.log(this.nome + " está comendo.");  
 }  
};  
  
// Função construtora para criar objetos Cachorro  
function Cachorro(nome, raca) {  
 this.nome = nome;  
 this.raca = raca;  
}  
  
// Definindo o protótipo de Cachorro para herdar de AnimalProto  
// Object.create cria um novo objeto com o protótipo especificado  
Cachorro.prototype = Object.create(AnimalProto);  
Cachorro.prototype.constructor = Cachorro; // Corrige o construtor  
  
// Adicionando um método específico para Cachorro  
Cachorro.prototype.latir = function() {  
 console.log(this.nome + " está latindo: Au au!");  
};  
  
const meuCachorro = new Cachorro("Rex", "Labrador");  
  
meuCachorro.fazerBarulho(); // Saída: Barulho genérico de animal.  
meuCachorro.comer(); // Saída: Rex está comendo.  
meuCachorro.latir(); // Saída: Rex está latindo: Au au!  
  
console.log(meuCachorro.\_\_proto\_\_ === Cachorro.prototype); // true  
console.log(Cachorro.prototype.\_\_proto\_\_ === AnimalProto); // true  
console.log(AnimalProto.\_\_proto\_\_ === Object.prototype); // true  
console.log(Object.prototype.\_\_proto\_\_ === null); // true

Neste exemplo:

* AnimalProto é um objeto simples que serve como protótipo para Cachorro.
* Cachorro.prototype é definido para herdar de AnimalProto usando Object.create(). Isso estabelece a cadeia de protótipos.
* Quando meuCachorro.fazerBarulho() é chamado, o JavaScript primeiro procura fazerBarulho em meuCachorro. Não encontrando, ele procura em Cachorro.prototype. Não encontrando, ele procura em AnimalProto, onde encontra e executa o método.

#### Prototipação com a sintaxe class (ES6+)

Como mencionado, a sintaxe class é uma abstração sobre o modelo de protótipos. O exemplo de herança com class que você já tem no material (class Animal e class Cachorro extends Animal) internamente utiliza a cadeia de protótipos para funcionar. Quando você define class Cachorro extends Animal, o JavaScript faz o seguinte por baixo dos panos:

* Define Cachorro.prototype para herdar de Animal.prototype.
* Garante que super() no construtor de Cachorro chame o construtor de Animal e configure corretamente o this.

// Relembrando o exemplo com 'class'  
class Animal {  
 constructor(nome) {  
 this.nome = nome;  
 }  
  
 fazerBarulho() {  
 console.log("Algum barulho...");  
 }  
}  
  
class Cachorro extends Animal {  
 constructor(nome, raca) {  
 super(nome);  
 this.raca = raca;  
 }  
  
 fazerBarulho() {  
 console.log("Au au!");  
 }  
  
 latir() {  
 console.log("O cachorro " + this.nome + " está latindo!");  
 }  
}  
  
const meuCachorroES6 = new Cachorro("Buddy", "Poodle");  
meuCachorroES6.fazerBarulho(); // Au au!  
meuCachorroES6.latir(); // O cachorro Buddy está latindo!  
  
// A cadeia de protótipos ainda está lá!  
console.log(meuCachorroES6.\_\_proto\_\_ === Cachorro.prototype); // true  
console.log(Cachorro.prototype.\_\_proto\_\_ === Animal.prototype); // true  
console.log(Animal.prototype.\_\_proto\_\_ === Object.prototype); // true

**Observações:**

* Entender a prototipação é crucial para dominar o JavaScript, mesmo usando a sintaxe class. Problemas como o uso incorreto de this em funções e a otimização de memória em aplicações JavaScript muitas vezes se relacionam diretamente com o modelo de protótipos.
* A prototipação permite uma herança mais flexível e dinâmica do que a herança baseada em classes, possibilitando a modificação de protótipos em tempo de execução.

**Documentação Oficial:**

* JavaScript: [Herança e cadeia de protótipos](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Inheritance_and_the_prototype_chain)
* JavaScript: [Classes](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Classes)

### Recursos Adicionais:

## 6. Introdução a Laravel, React Native e SQL

### 6.2. Laravel

Laravel é um framework PHP de código aberto, robusto e elegante, projetado para o desenvolvimento de aplicações web modernas. Ele segue o padrão de arquitetura Model-View-Controller (MVC) e oferece uma vasta gama de ferramentas e recursos que simplificam tarefas comuns de desenvolvimento, como roteamento, autenticação, ORM (Object-Relational Mapping), filas, e muito mais. A filosofia do Laravel é tornar o desenvolvimento web uma experiência agradável e produtiva.

#### Principais Conceitos e Comandos (Laravel)

##### Instalação e Configuração

Para começar com Laravel, você precisa ter PHP e Composer (gerenciador de dependências do PHP) instalados. O Laravel Installer é a maneira mais rápida de criar novos projetos.

* **Instalar Laravel Installer (globalmente):**
* **Criar um novo projeto Laravel:**
* **Iniciar o servidor de desenvolvimento:**

##### Artisan Console

Artisan é a interface de linha de comando incluída no Laravel. Ele fornece uma série de comandos úteis para o desenvolvimento da sua aplicação.

* **Listar todos os comandos Artisan disponíveis:**
* **Obter ajuda para um comando específico:**

##### Rotas (Routing)

As rotas definem como as requisições HTTP são tratadas pela sua aplicação. As rotas são definidas no arquivo routes/web.php (para rotas web) e routes/api.php (para APIs).

// routes/web.php  
  
use Illuminate\Support\Facades\Route;  
  
Route::get('/', function () {  
 return view('welcome');  
});  
  
Route::get('/saudacao/{nome}', function ($nome) {  
 return 'Olá, ' . $nome . '!';  
});  
  
Route::post('/contato', function () {  
 // Lógica para processar o formulário de contato  
 return 'Mensagem enviada!';  
});

##### Controladores (Controllers)

Controladores agrupam a lógica de requisição HTTP. Eles são criados usando o Artisan.

* **Criar um controlador:**
* **Exemplo de controlador (app/Http/Controllers/PostController.php):**
* **Associar rotas a controladores:**

##### Migrações (Migrations)

Migrações são como controle de versão para o seu banco de dados, permitindo que você defina e modifique a estrutura do banco de dados usando código PHP.

* **Criar uma migração:**
* **Executar migrações (criar tabelas no banco de dados):**
* **Reverter a última migração:**

##### Modelos (Models) e Eloquent ORM

Eloquent ORM é o ORM (Object-Relational Mapping) padrão do Laravel, que facilita a interação com o banco de dados usando objetos PHP.

* **Criar um modelo:**
* **Exemplo de modelo (app/Models/Post.php):**
* **Interagindo com o banco de dados usando Eloquent:**

##### Views (Blade)

Blade é o motor de template simples, mas poderoso, incluído no Laravel. As views são armazenadas em resources/views/.

* **Exemplo de view (resources/views/welcome.blade.php):**
* **Retornar uma view de uma rota ou controlador:**

**Observações:**

* Laravel é um ecossistema vasto. Esta seção aborda apenas os conceitos fundamentais para iniciar o desenvolvimento. Recursos como autenticação, autorização, filas, eventos, testes e pacotes são componentes poderosos que o Laravel oferece.

**Documentação Oficial:**

* Laravel: [Documentação Oficial](https://laravel.com/docs/)

### 6.3. React Native

React Native é um framework de código aberto, criado pelo Facebook, que permite desenvolver aplicações móveis nativas para iOS e Android usando JavaScript e React. Ao invés de renderizar componentes web (como no React para web), o React Native renderiza componentes nativos da interface do usuário, proporcionando uma experiência de usuário e performance próximas às de aplicações desenvolvidas com linguagens nativas (Swift/Objective-C para iOS, Java/Kotlin para Android).

#### Principais Conceitos e Comandos (React Native)

##### Instalação e Configuração

Existem duas maneiras principais de iniciar um projeto React Native:

1. **Expo Go (Recomendado para iniciantes e prototipagem rápida):** Permite desenvolver rapidamente sem a necessidade de configurar um ambiente de desenvolvimento nativo completo. Você escreve o código JavaScript e o Expo Go cuida da compilação e execução no seu dispositivo ou emulador.

* **Instalar Expo CLI (globalmente):**
* **Criar um novo projeto Expo:**
* **Iniciar o servidor de desenvolvimento Expo:**

1. **React Native CLI (Para projetos mais complexos e controle total sobre o código nativo):** Requer a configuração de ambientes de desenvolvimento para iOS (Xcode) e Android (Android Studio).

* **Instalar React Native CLI (globalmente):**
* **Criar um novo projeto React Native:**
* **Executar o app no iOS (requer Xcode e macOS):**
* **Executar o app no Android (requer Android Studio e emulador/dispositivo):**

##### Componentes Essenciais

React Native fornece um conjunto de componentes nativos que mapeiam para os elementos da UI nativa da plataforma. Alguns dos mais comuns incluem:

* **View**: O contêiner mais fundamental para construir a UI. Equivale a uma div no HTML.
* **Text**: Usado para exibir texto.
* **Image**: Usado para exibir imagens.
* **TextInput**: Um componente de entrada de texto.
* **Button**: Um componente de botão básico.
* **ScrollView**: Um contêiner que permite rolagem.
* **FlatList**: Um componente otimizado para exibir listas grandes de dados.
* **StyleSheet**: Usado para criar estilos para os componentes, similar ao CSS, mas com sintaxe JavaScript.

##### Exemplo Básico de Componente

Um componente funcional simples em React Native:

import React from 'react';  
import { View, Text, StyleSheet } from 'react-native';  
  
const MeuPrimeiroComponente = () => {  
 return (  
 <View style={styles.container}>  
 <Text style={styles.titulo}>Olá, React Native!</Text>  
 <Text style={styles.subtitulo}>Este é o meu primeiro aplicativo móvel.</Text>  
 </View>  
 );  
};  
  
const styles = StyleSheet.create({  
 container: {  
 flex: 1,  
 justifyContent: 'center',  
 alignItems: 'center',  
 backgroundColor: '#F5FCFF',  
 },  
 titulo: {  
 fontSize: 24,  
 textAlign: 'center',  
 margin: 10,  
 color: '#333333',  
 },  
 subtitulo: {  
 fontSize: 18,  
 textAlign: 'center',  
 color: '#666666',  
 },  
});  
  
export default MeuPrimeiroComponente;

##### Navegação

Para navegação entre telas, a biblioteca mais popular é o React Navigation.

* **Instalar React Navigation:**
* **Exemplo de Stack Navigator:**

##### Gerenciamento de Estado

Assim como no React para web, o gerenciamento de estado é crucial em React Native. As opções incluem:

* **useState**\*\* e **useContext** (Hooks do React):\*\* Para estado local e global simples.
* **Redux, Zustand, MobX:** Para gerenciamento de estado mais complexo em aplicações maiores.

##### Requisições de Rede

Para buscar dados de APIs, você pode usar a API fetch nativa do JavaScript ou bibliotecas como axios.

import React, { useEffect, useState } from 'react';  
import { View, Text, FlatList, ActivityIndicator } from 'react-native';  
  
const ListaDePosts = () => {  
 const [posts, setPosts] = useState([]);  
 const [loading, setLoading] = useState(true);  
  
 useEffect(() => {  
 fetch('https://jsonplaceholder.typicode.com/posts')  
 .then(response => response.json())  
 .then(data => {  
 setPosts(data);  
 setLoading(false);  
 })  
 .catch(error => {  
 console.error('Erro ao buscar posts:', error);  
 setLoading(false);  
 });  
 }, []);  
  
 if (loading) {  
 return (  
 <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>  
 <ActivityIndicator size="large" color="#0000ff" />  
 <Text>Carregando posts...</Text>  
 </View>  
 );  
 }  
  
 return (  
 <View style={{ flex: 1, paddingTop: 20 }}>  
 <FlatList  
 data={posts}  
 keyExtractor={item => item.id.toString()}  
 renderItem={({ item }) => (  
 <View style={{ padding: 10, borderBottomWidth: 1, borderBottomColor: '#ccc' }}>  
 <Text style={{ fontSize: 18, fontWeight: 'bold' }}>{item.title}</Text>  
 <Text>{item.body}</Text>  
 </View>  
 )}  
 />  
 </View>  
 );  
};  
  
export default ListaDePosts;

**Observações:**

* React Native é uma ferramenta poderosa para o desenvolvimento móvel, mas exige uma compreensão dos conceitos de React e, para projetos mais avançados, familiaridade com o ambiente de desenvolvimento nativo.
* A comunidade React Native é muito ativa, com muitas bibliotecas e recursos disponíveis para estender as funcionalidades do seu aplicativo.

**Documentação Oficial:**

* React Native: [Documentação Oficial](https://reactnative.dev/docs/)
* Expo: [Documentação Oficial](https://docs.expo.dev/)
* React Navigation: [Documentação Oficial](https://reactnavigation.org/)

### 6.1. SQL Portátil: Comandos Essenciais

SQL (Structured Query Language) é a linguagem universal para interagir com bancos de dados relacionais. Embora existam dialetos específicos para cada sistema de gerenciamento de banco de dados (SGBD) como MySQL, PostgreSQL, SQL Server, etc., a maioria dos comandos básicos segue o padrão ANSI SQL, o que os torna portáteis entre diferentes sistemas. Esta seção foca nesses comandos universais.

#### Linguagem de Definição de Dados (DDL)

DDL é usada para definir e gerenciar a estrutura do banco de dados e seus objetos, como tabelas.

| Comando | Descrição | Exemplo de Uso | Observações |
| --- | --- | --- | --- |
| CREATE TABLE | Cria uma nova tabela no banco de dados. | CREATE TABLE usuarios (id INT PRIMARY KEY, nome VARCHAR(255) NOT NULL, email VARCHAR(255) UNIQUE); | Os tipos de dados (ex: INT, VARCHAR) são amplamente suportados, mas podem ter nomes ou limites diferentes em alguns SGBDs. AUTO\_INCREMENT (MySQL) é SERIAL em PostgreSQL e IDENTITY em SQL Server. |
| ALTER TABLE | Modifica a estrutura de uma tabela existente. | ALTER TABLE usuarios ADD data\_nascimento DATE; ALTER TABLE usuarios DROP COLUMN email; | A sintaxe para adicionar, remover ou modificar colunas é bastante padronizada. |
| DROP TABLE | Exclui permanentemente uma tabela e todos os seus dados. | DROP TABLE usuarios; | Ação irreversível. Use com cuidado. |
| CREATE INDEX | Cria um índice em uma ou mais colunas para acelerar as consultas. | CREATE INDEX idx\_nome ON usuarios (nome); | Essencial para otimização de performance em tabelas grandes. |

#### Linguagem de Manipulação de Dados (DML)

DML é usada para gerenciar os dados dentro das tabelas.

| Comando | Descrição | Exemplo de Uso |
| --- | --- | --- |
| INSERT INTO | Adiciona uma ou mais novas linhas (registros) a uma tabela. | INSERT INTO usuarios (id, nome, email) VALUES (1, 'Ana Silva', 'ana.silva@example.com'); |
| UPDATE | Modifica registros existentes em uma tabela. | UPDATE usuarios SET email = 'ana.s@newdomain.com' WHERE id = 1; |
| DELETE | Remove registros de uma tabela. | DELETE FROM usuarios WHERE id = 1; |

#### Linguagem de Consulta de Dados (DQL)

DQL é usada para consultar e recuperar dados do banco de dados. O comando SELECT é o principal componente da DQL.

| Cláusula/Operador | Descrição | Exemplo de Uso com SELECT |
| --- | --- | --- |
| SELECT | Recupera dados de uma ou mais tabelas. | SELECT nome, email FROM usuarios; SELECT \* FROM usuarios; |
| FROM | Especifica a tabela da qual os dados serão recuperados. | SELECT nome FROM usuarios; |
| WHERE | Filtra os registros com base em uma condição. | SELECT \* FROM usuarios WHERE idade > 18; |
| ORDER BY | Ordena os resultados com base em uma ou mais colunas. | SELECT \* FROM usuarios ORDER BY nome ASC; (ASC: ascendente, DESC: descendente) |
| GROUP BY | Agrupa linhas que têm os mesmos valores em colunas especificadas em linhas de resumo. | SELECT pais, COUNT(id) FROM usuarios GROUP BY pais; |
| HAVING | Filtra os resultados de um GROUP BY com base em uma condição. | SELECT pais, COUNT(id) FROM usuarios GROUP BY pais HAVING COUNT(id) > 10; |

#### Funções Agregadas Padrão

Funções que realizam um cálculo em um conjunto de valores e retornam um único valor.

| Função | Descrição | Exemplo de Uso |
| --- | --- | --- |
| COUNT() | Conta o número de linhas. | SELECT COUNT(\*) FROM usuarios; |
| SUM() | Soma os valores de uma coluna numérica. | SELECT SUM(salario) FROM funcionarios; |
| AVG() | Calcula a média dos valores de uma coluna numérica. | SELECT AVG(idade) FROM usuarios; |
| MIN() | Retorna o menor valor de uma coluna. | SELECT MIN(preco) FROM produtos; |
| MAX() | Retorna o maior valor de uma coluna. | SELECT MAX(preco) FROM produtos; |

#### Junções (JOINs)

Combinam linhas de duas ou mais tabelas com base em uma coluna relacionada.

| Tipo de JOIN | Descrição | Exemplo de Uso |
| --- | --- | --- |
| INNER JOIN | Retorna registros que têm valores correspondentes em ambas as tabelas. | SELECT u.nome, p.nome\_produto FROM usuarios u INNER JOIN pedidos p ON u.id = p.usuario\_id; |
| LEFT JOIN | Retorna todos os registros da tabela da esquerda e os registros correspondentes da tabela da direita. | SELECT u.nome, p.nome\_produto FROM usuarios u LEFT JOIN pedidos p ON u.id = p.usuario\_id; |
| RIGHT JOIN | Retorna todos os registros da tabela da direita e os registros correspondentes da tabela da esquerda. | SELECT u.nome, p.nome\_produto FROM usuarios u RIGHT JOIN pedidos p ON u.id = p.usuario\_id; |
| FULL OUTER JOIN | Retorna todos os registros quando há uma correspondência na tabela da esquerda ou da direita. | SELECT u.nome, p.nome\_produto FROM usuarios u FULL OUTER JOIN pedidos p ON u.id = p.usuario\_id; |

**Observações:**

* A sintaxe apresentada é amplamente compatível com a maioria dos SGBDs modernos. No entanto, sempre consulte a documentação específica do seu SGBD para funcionalidades avançadas ou tipos de dados específicos.

**Documentação de Referência:**

* [Padrões SQL (ANSI/ISO)](https://www.iso.org/standard/84340.html)
* [W3Schools SQL Tutorial](https://www.w3schools.com/sql/)