* **Versões do PHP**
  + **1995: PHP 1.0**
    - Personal Home Page
    - Funcionalidades simples para um site básico.
    - Não tinha características de linguagem em si.
  + **1997: PHP 2.0**
    - Oficialmente nomeado PHP/FI 2.0.
    - Linguagem standalone.
    - Recursos limitados.
  + **1998: PHP 3.0**
    - Primeira versão colaborativa.
    - Características mais relacionadas a uma linguagem.
    - Extension API.
  + **2000: PHP 4.0**
    - Zend Engine 1.0.
    - Melhoria de performance.
    - Aumento da modularização.
    - Uso de super globais $\_GET, $\_POST, $\_SESSION, etc.
  + **2004: PHP 5.0**
    - Zend Engine 2.0.
    - Primeira versão com Orientação a Objetos.
    - PHP Data Objects (PDO).
    - Operador de exponenciação.
    - Suporte a JSON.
    - Namespaces, closures, garbage collection, exception handling
    - Novo depurador (phpdbg)
  + **2005: PHP 6.0**
    - Aumento da segurança e performance.
    - Suporte nativo a Unicode.
    - Até existiram algumas versões.
    - Nunca recebeu uma release oficial.
  + **2015: PHP 7.0**
    - Zend Engine 3.0.
    - Performance até 9 vezes maior.
    - Escape de códigos Unicode
    - Operador null coalescing ??
    - Declaração escalar
    - Operador spaceship <=>
    - Classes anônimas.
    - Retorno de void.
    - Libsodium
    - Foreign function interface
  + **2020: PHP 8.0**
    - Zend Engine 4.0.
    - Just-In-Time compilation.
    - Locale-Independent number convertions.
    - Named arguments.
    - Expressão match.
    - Operador nullsafe.
    - Enumerations.
    - Readonly properties.
    - Fibers.
* **Regras para nomes dos identificadores:**
  + Variáveis sempre começam com o símbolo $
  + O segundo pode ser letra ou o símbolo \_
  + Aceita caracteres [a-z], [A-Z], [0-9] e [\_]
  + Aceita caracteres da tabela ASCII a partir de 128
  + Aceita caracteres acentuados como à, õ, ç
  + A linguagem é case sensitive em relação aos nomes
  + Nomes especiais como $this não podem ser usados
* **Recomendações para dar nomes:**
  + Tente dar nomes claros e de fácil identificação
  + Evite nomes muito curtos ou muito longos
  + Defina um padrão e siga em todo o projeto
  + Para variáveis, dê preferência a letras minúsculas
  + Para constantes, dê preferência a letras maiúsculas
  + Use camelCase para métodos e atributos
  + Use SNAKE\_CASE para nomear constantes
* **Categorias dos tipos primitivos:**
  + **Escalares:**
    - **String:** Sequência de letras, números e símbolos, sempre representadas entre aspas.
    - **Int ou integer:** Um valor numérico inteiro, aquele que vem sem a parte de cima.
    - **Float, double ou real:** O valor numérico real, que vem com a parte decimal, depois do ponto flutuante.
    - **Bool ou boolean:** Um valor lógico ou Booleano, que aceita apenas os valores verdadeiro ou falso (true ou false).

0x = hexadecimal notation

0b = binary notation

0 octal notation

(type) <value> = Type Coercion

* + **Compostos:**
    - Array
    - Object
  + **Especiais:**
    - Null
    - Resource
    - Callabe
    - Mixed
* **4 formatos de Strings:**
  + **Double quoted** -> “asdf” -> **existe a interpretação**
    - “PHP \u{1F418}” -> PHP<símbolo de \u{1F418}>
  + **Single quoted** -> ‘asdf’ -> **não existe a interpretação**
    - ‘PHP \u{1F418}’ -> PHP\u{1F418}

A exceção é quando se passa uma **constante** ao invés de uma **variável**. Nesse caso, não há interpretação na área.

Concatenation operator -> “asdf” . “ghjk”

* + **Heredoc** -> bloco de string -> as palavras de início e fim de bloco de string devem ser a mesma palavra, e **existe a interpretação**, assim como em **double quoted**

echo <<< [palavra de início de bloco]

<conteúdo do bloco de string>

[palavra de fim de bloco];

* + **Nowdoc** -> bloco de string -> as palavras de início e fim de bloco de string devem ser a mesma palavra, e **não existe a interpretação**, assim como em **single quoted**

echo <<< ‘[palavra de início de bloco]’

<conteúdo do bloco de string>

[palavra de fim de bloco];

* **Operadores aritméticos:**
  + Adição: +
  + Subtração: -
  + Multiplicação: \*
  + Divisão: /
  + Módulo: %
  + Exponenciação: \*\*
* **Precedência:**
  + 1º \*\*
  + 2º \* / %
  + 3º + -
* **Funções aritméticas**
  + abs(<num>) = remove sinal do número
  + base\_convert(<num>, <base\_origem>, <base\_final>) = converte o número para uma base numérica diferente
  + ceil(<num>), floor(<num>), round(<num>) = arredonda o número sempre para cima (ceil()), sempre para baixo (floor()) ou segue as normas para arredondamento (round())
  + hypot(<num>) = calcula a hipotenusa de um número
  + intdiv(<num1>, <num2>) = retorna o quociente inteiro de uma divisão de num1 dividido por num2
  + min(<num1>, ..., <numN>) = retorna o valor mínimo entre num1 e numN
  + max(<num1>, ..., <numN>) = retorna o valor máximo entre num1 e numN
  + pi() = retorna o valor de pi
  + pow(<num1>, <num2>) = eleva num1 a num2
  + sqrt(<num>) = calcula a raíz quadrada de num
  + sin(<num>), cos(<num>), tan(<num>) = retorna o seno, coseno e tangente de <num>