Apunte: Introducción a PHP

Introducción a la Programación (IP) Tecnicatura en Desarrollo Web

20 agosto, 2023

3.1. Introducción

Esta sección se incluye un resumen sobre algunos conceptos de PHP que son elementales para comenzar a programar con el lenguaje. Se describe como especificar variables, constantes, tipos de dato, operadores y operandos, expresiones, el concepto de **asignación** en PHP. Es una sección que muestra como los conceptos que fueron desarrollados en la sección anterior se aplican a un lenguaje de programación (en este caso PHP).

Adicionalmente se resume en el apunte la sintáxis asociada a estos conceptos, características de entrada-salida, y la estructura básica de un programa PHP.

En IP utilizaremos el lenguaje de programación php para realizar las implementaciones de nuestros algoritmos. Utilizaremos Visual Studio Code, un entorno de desarrollo en el cual es posible utilizar PHP, para programar. Es posible seguir los siguientes videos para descargar ambos softwares, instalarlos en Ubuntu o Windows y finalmente ejecutar una aplicación hola mundo!

3.1.1. Entorno de desarrollo en Ubuntu

Instalación Intérprete PHP en Ubuntu 19

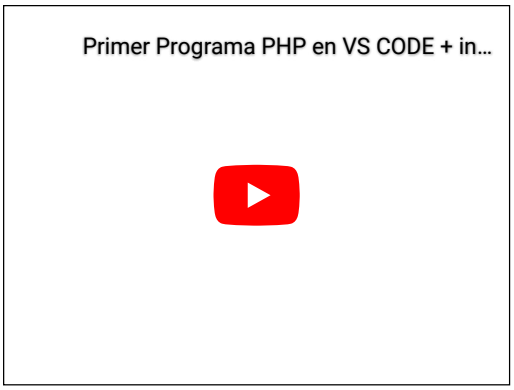
Instalación VSCode en Ubuntu

3.1.2. Entorno de Desarrollo en Windows

Instalacion PHP en Windows 10

Visual Studio Code | Instalación de Vis...

3.1.3. Extensiones VSCode



3.2. Construcción de Programas

Cuando nos piden que hagamos un programa debemos seguir una cantidad de pasos para asegurarnos de que tendremos éxito en la tarea. La acción irreflexiva (me piden algo, me siento frente a la computadora y escribo rápidamente sin pensar lo que me parece que es la solución) no constituye una actitud profesional (e ingenieril) de resolución de problemas. Toda construcción tiene que seguir una metodología, un protocolo de desarrollo dado.

Existen muchas metodologías para construir programas, pero en este curso aplicaremos una metodología sencilla, que es adecuada para la construcción de programas pequeños, y que se puede resumir en los siguientes pasos:

1. **Analizar el problema.** Entender profundamente cuál es el problema que se trata de resolver, incluyendo el contexto en el cual se usará.

> Una vez analizado el problema, asentar el análisis por escrito.

2. **Especificar la solución.** Este es el punto en el cual se describe qué debe hacer el programa, sin importar él cómo. En el caso de los problemas sencillos que abordaremos, deberemos decidir cuáles son los datos de entrada que se nos darán, cuáles son las salidas que debemos producir, y cuál es la relación entre todos ellos.

> Al especificar el problema a resolver, documentar la especificación por escrito.

3. **Diseñar la solución.** Este es el punto en el cuál atacamos el cómo vamos a resolver el problema, cuáles ´son los algoritmos y las estructuras de datos que usaremos. Analizamos posibles variantes, y las decisiones las tomamos usando como dato de la realidad el contexto en el que se aplicará la solución, y los costos asociados a cada diseño.

> Luego de diseñar la solución, asentar por escrito el diseño, asegurándonos de que esté completo.

4. **Implementar el diseño.** Traducir a un lenguaje de programación (en nuestro caso, y por el momento, PHP) el diseño que elegimos en el punto anterior.

> La implementación también se debe documentar, con comentarios dentro y fuera del código, al respecto de qué hace el programa, cómo lo hace y por qué lo hace de esa forma.

5. **Probar el programa.** Diseñar un conjunto de pruebas para probar cada una de sus partes por separado, y también la correcta integración entre ellas. Utilizar el depurador como instrumento para descubrir dónde se producen ciertos errores.

> Al ejecutar las pruebas, documentar los resultados obtenidos.

6. **Mantener el programa.** Realizar los cambios en respuesta a nuevas demandas.

> Cuando se realicen cambios, es necesario documentar el análisis, la especificación, el diseño, la implementación y las pruebas que surjan para llevar estos cambios a cabo.

3.2.1. Una guía para el diseño

En su artículo “How to program it”, Simon Thompson plantea algunas preguntas a sus alumnos que son muy útiles para la etapa de diseño:

- ¿Han visto este problema antes, aunque sea de manera ligeramente diferente?
- ¿Conocen un problema relacionado? ¿Conocen un programa que puede ser ´util?
- Fíjense en la especificación. Traten de encontrar un problema que les resulte familiar, que tenga la misma especificación o una parecida.
- Acá hay un problema relacionado con el que ustedes tienen y que ya fue resuelto. ¿Lo pueden usar? ¿Pueden usar sus resultados? ¿Pueden usar sus métodos? ¿Pueden agregarle alguna parte auxiliar a ese programa del que ya disponen?
- Si no pueden resolver el problema propuesto, traten de resolver uno relacionado. ¿Pueden imaginarse uno relacionado que sea más fácil de resolver? ¿Uno más general? ¿Uno más específico? ¿Un problema análogo? ¿Pueden resolver una parte del problema? ¿Pueden sacar algo ´util de los datos de entrada? ¿Pueden pensar qué información es ´util para calcular las salidas? ¿De qué manera se pueden manipular las entradas y las salidas de modo tal que estén “más cerca” unas de las otras?
- ¿Usaron todos los datos de entrada? ¿Usaron las condiciones especiales sobre los datos de entrada que aparecen en el enunciado? ¿Han tenido en cuenta todos los requisitos que se enuncian en la especificación?

3.2.2. Realizando un programa sencillo

Al leer un artículo en una revista norteamericana que contiene información de longitudes expresadas en millas, pies y pulgadas, queremos poder convertir esas distancias de modo que sean fáciles de entender. Para ello, decidimos escribir un programa que convierta las longitudes del sistema inglés al sistema métrico decimal. Antes de comenzar a programar, utilizamos la guía de la sección anterior, para analizar, especificar, diseñar, implementar y probar el problema.

1. **Análisis del problema.** En este caso el problema es sencillo: nos dan un valor expresado en millas, pies y pulgadas y queremos transformarlo en un valor en el sistema métrico decimal. Sin embargo hay varias respuestas posibles, porque no hemos fijado en qué unidad queremos el resultado. Supongamos que decidimos que queremos expresar todo en metros.

2. **Especificación.** Debemos establecer la relación entre los datos de entrada y los datos de salida. Ante todo debemos averiguar los valores para la conversión de las unidades básicas. Buscando en Internet y encontramos la siguiente tabla: - 1 milla = 1.609344 km

- 1 pie = 30.48 cm
- 1 pulgada = 2.54 cm

> A lo largo de todo el curso usaremos punto decimal, en lugar de coma decimal, para representar valores no enteros, dado que esa es la notación que utiliza PHP.

La tabla obtenida no traduce las longitudes a metros. La manipulamos para llevar todo a metros: - 1 milla = 1609.344 m - 1 pie = 0.3048 m - 1 pulgada = 0.0254 m

Si una longitud se expresa como L millas, F pies y P pulgadas, su conversión a metros se calculará como M = 1609.344 * L + 0.3048 * F + 0.0254 * P Hemos especificado el problema. Pasamos entonces a la próxima etapa.

3. **Diseño.** La estructura de este programa es sencilla: leer los datos de entrada, calcular la solución, mostrar el resultado, o Entrada-Cálculo-Salida. Antes de escribir el programa, escribiremos en pseudocódigo (un castellano preciso que se usa para describir lo que hace un programa) una descripción del mismo:

Leer cuántas millas tiene la longitud dada (y referenciarlo con la variable millas) Leer cuántos pies tiene la longitud dada (y referenciarlo con la variable pies) Leer cuántas pulgadas tiene la longitud dada (y referenciarlo con la variable pulgadas) Calcular metros = 1609.344 * millas + 0.3048 * pies + 0.0254 * pulgadas Mostrar por pantalla la variable metros

```
PROGRAMA ConvertirSistemaMetrico
  (*Suma tres longitudes en millas, pies y pulgadas convirtiéndolas en metros*)
  REAL millas, pies, pulgadas, metros
  ESCRIBIR(“¿Cuántas millas?: ”)
  LEER(millas)
  ESCRIBIR(“¿Cuántos pies?: ”)
  LEER(pies)
  ESCRIBIR(“¿Cuántas pulgadas?: ”)
  LEER(pulgadas)
  metros ← 1609.344 * millas + 0.3048 * pies + 0.0254 * pulgadas
  ESCRIBIR(“la longitud es de” + metros + “ metros ”)
FIN PROGRAMA
```

4. **Implementación.** Ahora estamos en condiciones de traducir este pseudocódigo a un programa en lenguaje PHP:

```
<?php
  /* Suma tres longitudes en millas , pies y pulgadas convirtiéndolas en metros */
  /* REAL $millas ,$pies , $pulgadas , $metros */
  echo " Convierte medidas inglesas a sistema metrico ";

  echo " ¿Cuántas millas ?:";
  $millas = trim ( fgets ( STDIN ));

  echo " ¿Cuántos pies ?:";
  $pies = trim ( fgets ( STDIN ));

  echo " ¿Cuántas pulgadas ?:";
  $pulgadas = trim ( fgets ( STDIN ));

  $metros = 1609.344 * $millas + 0.3048 * $pies + 0.0254 * $pulgadas ;

  echo "La longitud es de " . $metros . " metros " ;
? >
```

5. **Prueba.** Probaremos el programa para valores para los que conocemos la solución:

- 1 milla, 0 pies, 0 pulgadas.
- 0 millas, 1 pie, 0 pulgada.
- 0 millas, 0 pies, 1 pulgada.

En la sección anterior hicimos hincapié en la necesidad de documentar todo el proceso de desarrollo. En este ejemplo la documentación completa del proceso lo constituye todo lo escrito en esta sección. Al entregar un ejercicio, se deberá presentar el desarrollo completo con todas las etapas, desde el análisis hasta las pruebas (y el mantenimiento, si hubo cambios).

3.3. Identificadores

Utilizamos nombres significativos para denominar archivos de programas, para denominar variables, constantes y funciones. Todos esos nombres se denominan identificadores y PHP tiene reglas sobre qué es un identificador válido y qué no lo es. Un identificador de una variable comienza con el símbolo $ y luego sigue con una secuencia de letras, números y guiones bajos. Los espacios no están permitidos en el nombre.

Los siguientes son ejemplos de identificadores válidos:

- $hola
- $hota12t
- $_hola
- $Hola

PHP distingue mayúsculas de minúsculas (en inglés case sensitive), así que *Hola* es identificador y *hola* es otro identificador. Los siguientes son todos identificadores inválidos en PHP:

- $hola 12t
- $$hota12t
- $hola%
- $Hola*9
- hola$

3.2.1 Palabras Reservadas (keywords)

PHP reserva palabras para describir estructura del programa, y no permite que esas palabras se utilicen como identificadores. Cuando en un programa nos encontramos con que un nombre no es admitido pese a que su formato es válido, seguramente se trata de una de las palabras de la tabla mostrada a continuación, a la que denominaremos **palabras reservadas de PHP**.

Palabras Reservadas de PHP

__halt_compiler() (https://www.php.net/manual/en/function.halt-compiler.php)	abstract (https://www.php.net/manual/en/language.oop5.abstract.php)	and (https://www.php.net/manual/en/language.operator
break (https://www.php.net/manual/en/control-structures.break.php)	callable (https://www.php.net/manual/en/language.types.callable.php)	case (https://www.php.net/manual/en/control-structures.switch.php)
clone (https://www.php.net/manual/en/language.oop5.cloning.php)	const (https://www.php.net/manual/en/language.oop5.constants.php)	continue (https://www.php.net/manual/en/control-structures.continue.php)
die() (https://www.php.net/manual/en/function.die.php)	do (https://www.php.net/manual/en/control-structures.do.while.php)	echo (https://www.php.net/manual/en/function.ech
empty() (https://www.php.net/manual/en/function.empty.php)	enddeclare (https://www.php.net/manual/en/control-structures.declare.php)	endfor (https://www.php.net/manual/en/control-structures.alternative-syntax.php)
endswitch (https://www.php.net/manual/en/control-structures.alternative-syntax.php)	endwhile (https://www.php.net/manual/en/control-structures.alternative-syntax.php)	eval() (https://www.php.net/manual/en/function.eva
final (https://www.php.net/manual/en/language.oop5.final.php)	finally (https://www.php.net/manual/en/language.exceptions.php)	fn (https://www.php.net/manual/en/functions.arrow PHP 7.4)
function (https://www.php.net/manual/en/functions.user-defined.php)	global (https://www.php.net/manual/en/language.variables.scope.php)	goto (https://www.php.net/manual/en/control-structures.goto.php)
include (https://www.php.net/manual/en/function.include.php)	include_once (https://www.php.net/manual/en/function.include-once.php)	instanceof (https://www.php.net/manual/en/language.operator
isset() (https://www.php.net/manual/en/function.isset.php)	list() (https://www.php.net/manual/en/function.list.php)	match (https://www.php.net/manual/en/control-structures.match.php) (as of PHP 8.0)
or (https://www.php.net/manual/en/language.operators.logical.php)	print (https://www.php.net/manual/en/function.print.php)	private (https://www.php.net/manual/en/language.oop5.vis
readonly (language.oop5.properties.php#language.oop5.properties.readonly-properties) (as of PHP 8.1.0) *	require (https://www.php.net/manual/en/function.require.php)	require_once (https://www.php.net/manual/en/func once.php)
switch (https://www.php.net/manual/en/control-structures.switch.php)	throw (https://www.php.net/manual/en/language.exceptions.php)	trait (https://www.php.net/manual/en/language.oop
use (https://www.php.net/manual/en/language.namespaces.php)	var (https://www.php.net/manual/en/language.oop5.properties.php)	while (https://www.php.net/manual/en/control-structures.while.php)
yield from (https://www.php.net/manual/en/language.generators.syntax.php#control-structures.yield.from)		

3.3.1 Variable y Constantes

En PHP las variables adquieren un tipo a partir de su asignación. Las variables pueden almacenar datos de un tipo primitivo (o un tipo clase), y su valor puede cambiar durante la ejecución del programa tantas veces como sea necesario, además pueden aplicarse ciertos operaciones dependiendo del tipo de las variables.

Convención: las variables comienzan con minúsculas y las constantes van todas en mayúsculas.

Ejemplos de identificadores de variables: miNombre, jugadoresDeFutball, juan, notaFinal, promedio.

Ejemplos de identificadores de constantes: VALOR_PI, TASA_ANUAL, INCREMENTO

3.3.1.1 Convención para identificadores

En programación, una convención de nombres es un conjunto de reglas para la elección de la secuencia de caracteres que se utilice para identificadores que denoten variables, tipos, funciones y otras entidades en el código fuente y la documentación.

Algunas de las razones para utilizar una convención de nombres (en lugar de permitir a los programadores elegir cualquier secuencia de caracteres) son:

- reducir el esfuerzo necesario para leer y entender el código fuente;
- mejorar la apariencia del código fuente (por ejemplo, al no permitir nombres excesivamente largos o abreviaturas poco claras).

> Convención: Para definir variables cuyos nombres son compuestos utilizaremos la notación **infixCaps**, la cual permite yuxtaponer dos palabras, comenzando la segunda palabra con su inicial en mayúscula.

PHP es un lenguaje **Case Sensitive**, es decir que es sensible a las minúsculas y mayúsculas. En pocas palabras, para PHP la variable contador es distinta a la variable Contador.

En PHP las constantes son variables que a las que se les aplica el modificador **CONST**. El valor de una variable declarada como final no puede cambiar durante la ejecución de un programa.

Ejemplo:

```
CONST $pi = 3.14159;
```

CONST $pi = 3.14159;

3.4. Tipos de Datos

Un tipo de datos define el conjunto de valores que puede tomar una variable y qué operaciones se pueden realizar con ella (comportamiento). El tipo de dato de una variable define cuáles son los valores que se pueden almacenar en la variable, por ejemplo en el caso de los números: enteros, reales, etc. Si una variable es de tipo entero, sólo podrá almacenar números enteros y no así reales, y tendrá definidas ciertas operaciones para manipular dichos valores. En cambio una variable de tipo real podrá almacenar un número entero, pues

los enteros también son reales, y tendrá definido un conjunto de operaciones que podrán aplicarse.

Los tipos de datos primitivos definidos por el lenguaje de programación son atómicos, es decir que permiten almacenar un solo valor en cada variable.

Grupo	Tipo de Datos	Descripción
REAL	float	Son números que tienen punto flotante, es decir tienen decimales.
ENTERO	integer	Son números que no tienen parte decimal.
LOGICO	boolean	Solo pueden tomar dos valores **true** o **false**.
CARACTER	string	Letras, números, símbolos especiales que van entre comillas simples
TEXTO	string	Son cadenas de caracteres.

Existen lenguajes que obligan al programador a declarar de qué tipo de dato serán las variables al momento de especificar el algoritmo. En este caso se habla de **lenguajes con tipos estáticos**, ya que se determina el tipo de todas las expresiones antes de la ejecución del programa (típicamente al compilar). Este es el caso de lenguajes como Pascal, Java, C, C++.

En Lenguajes como PHP, Python, JavaScript, el programador no debe declarar el tipo de las variables, ya que el tipo se establece durante la ejecución del programa. En este caso se habla de **lenguajes con tipos dinámicos**. En estos lenguajes una variable puede asumir distintos tipos durante la ejecución del programa, pero lo recomendado para que el código fuente se pueda interpretar, es tratar de mantener el tipo de las variables asignándoles expresiones cuyo resultado sea del tipo de la variable

Los tipos de datos del lenguaje PHP se pueden consultar en: http://php.net/manual/es/language.types.intro.php (http://php.net/manual/es/language.types.intro.php)

3.5. Operadores y Operandos

Los operadores son símbolos que representan algún tipo de operación sobre dos operandos (salvo los operadores unarios que actúan sobre un solo operando) y arrojan un resultado. Se deben considerar los tipos de los operandos para determinar cuales operadores son aplicables. Por ejemplo que si los operandos son números el operador deberá ser un operador aritmético.

En PHP existen varios tipos de operadores, los cuales representan operaciones que se realizan sobre operandos, y son listados en la siguiente Tabla:

Tipo de Operadores	Aritméticos	Relacionales	Lógicos
Operadores	* Multiplicación \ División % Módulo-Residuo + Adición - Diferencia	> mayor que < menor que >= mayor igual que <= menor igual que == igual a (comparación) ! = distinto a	&& and || or ^ xor ! not

3.6. Expresiones

Una **expresión** es todo aquello que se puede evaluar, es decir que lanza un resultado. Las siguientes son expresiones:

1. Variables
2. Constantes
3. Operando Operador Operando
4. Expresión Operador Expresión

Considerando los tipos de operadores, podemos decir que existen expresiones de tipo aritméticas, relacionales y lógicas, es decir expresiones que al ser evaluadas retornan un resultado de un cierto tipo. En cada caso tanto los operadores como los operandos deben ser compatibles, de lo contrario no se podría evaluar. Por ejemplo si intentamos sumar dos valores que no son numéricos, esta sería una expresión incorrecta que no se podría evaluar.

3.7. Asignación

El operador de asignación es aquel que permite que la evaluación de una expresión se almacene en una variable. El tipo de la expresión debe ser de tipo compatible con el tipo de la variable. Se representa con el símbolo de igualdad '=', pero no se debe confundir con el operador de comparación (en PHP '==').

3.8. Entrada Salida

Al hablar de entrada y salida nos referimos al ingreso de datos por parte del usuario y salida de la computadora por ejemplo en la pantalla al usuario.

La siguiente tabla describe ejemplos de entrada y salida de variables de distinto tipo:

Primitiva Pseudocódigo	PHP
Entrada: Mostrar mensajes por pantalla ESCRIBIR()	echo
Salida: Leer datos del usuario LEER()	trim(fgets(STDIN))

3.8.1 Ejemplos

3.8.1.1 Instrucción de Salida

```
ALGORITMO saludo() RETORNA ⌀
  (* alg. que saluda a amigos  *)
  ESCRIBIR(" Hola amigos !")
  ESCRIBIR(" Estoy programando en PHP")
FIN ALGORITMO saludo
```

```
<?php
 /* Alg. que saluda a amigos  */
 echo " Hola amigos !";
 echo " Estoy programando en PHP";
?>
```

```
##  Hola amigos ! Estoy programando en PHP
```

3.8.1.2 Instrucción de Salida con salto de linea

```
ALGORITMO saludo() RETORNA ⌀
  (* alg. que saluda a amigos  *)
  ESCRIBIR(" Hola amigos !") (* con salto *)
  ESCRIBIR(" Estoy programando en PHP")
FIN ALGORITMO saludo
```

```
<?php
 /* Alg. que saluda a amigos  */
 echo " Hola amigos ! \n";
 echo " Estoy programando en PHP";
?>
```

```
##  Hola amigos !
##  Estoy programando en PHP
```

3.8.1.2 Instrucción de Entrada

```
ALGORITMO saludo() RETORNA ⌀
  (* alg. que lee un nombre y lo saluda  *)
  TEXTO $nombre
  ESCRIBIR("Ingrese un nombre")
  LEER($nombre)
  ESCRIBIR(" Hola" + $nombre + "!")
  ESCRIBIR(" Estoy programando en PHP")
FIN ALGORITMO saludo
```

```
<?php
  /* Alg. que lee un nombre y lo saluda  */
  echo " Ingrese un nombre : \n" ;
  $nombre = trim(fgets(STDIN));

  echo " Hola ". $nombre ."!";
  echo " Estoy programando en PHP";
?>
```

```
## Ingrese un nombre:
## Ana
## Hola Ana !
## Estoy programando en PHP";
```

3.9 Expresiones

Una expresión es una porción de código PHP que produce o calcula un valor (resultado).

- Un valor es una expresión (de hecho es la expresión más sencilla). Por ejemplo el resultado de la expresión 111 es precisamente el n´umero 111.
- Una variable es una expresión, y el valor que produce es el que tiene asociado en el estado (si x ← 5 en el estado, entonces el resultado de la expresión x es el número 5).
- Usamos operaciones para combinar expresiones y construir expresiones más complejas:
- Si x tiene asignado el valor 5 , x + 1 es una expresión cuyo resultado es 6.
- Si en el estado millas ← 1, pies ← 0 y pulgadas ← 0, entonces 1609.344 * millas + 0.3048 * pies + 0.0254
- pulgadas es una expresión cuyo resultado es 1609.344.
- Se pueden usar paréntesis para indicar un orden de evaluación: ((b * b) - (4 * a * c)) / (x / y)
- Igual que en la matemática, si no hay paréntesis en la expresión primero se agrupan las exponenciaciones, luego los productos y cocientes, y luego las sumas y restas.
- Si x e y son n´umeros enteros, entonces x % y se calcula como el resto de la división entera entre x e y: Si x se refiere al valor 12 e y se refiere al valor 9 entonces x % y se refiere al valor 3.

> Los números pueden ser tanto enteros (111, -24), como reales (12.5, 12.0, -12.5). Dentro de la computadora se representan de manera diferente, y se comportan de manera diferente frente a las operaciones.

Conocemos también dos expresiones muy particulares:

- trim(fgets(STDIN)) devuelve el valor ingresado por teclado tal como se lo digita
- echo visualiza las salidas de la aplicación.
- **Ejercicio A:** Aplicando las reglas matemáticas de asociatividad, decidir cuáles de las siguientes expresiones son iguales entre sí:

1. ((b * b) - (4 * a * c)) / (2 * a)
2. (b * b - 4 * a * c) / (2 * a)
3. b * b - 4 * a * c / 2 * a
4. (b * b) - (4 * a * c / 2 * a)
5. 1 / 2 * b
6. b / 2

- **Ejercicio B.** Implementar en PHP un algoritmo que realice lo siguiente: darle a a, b y c los valores 10, 100 y 1000 respectivamente y evaluar las expresiones del ejercicio anterior.
- **Ejercicio C.** Implementar en PHP un algoritmo que realice lo siguiente: darle a a, b y c los valores 10.0, 100.0 y 1000.0 respectivamente y evaluar las expresiones del punto anterior.

3.10. Estructura de un Programa

Todo programa tiene una parte principal que dirige el funcionamiento del mismo. Utilizaremos el siguiente esquema general para definir todos nuestros programas.

```
ALGORITMO calcularPerimetro() RETORNA ⌀
  (*alg. que lee datos de un rectangulo,
  valores enteros mayores a 0, calc. perimetro *)
  REAL ladoMenor, ladoMayor, perimetro
  ESCRIBIR(”Ingrese el valor del lado menor”)
  LEER(ladoMenor)
  ESCRIBIR(”Ingrese el valor del lado mayor”)
  LEER(ladoMayor)
  perimetro ← 2*ladoMenor + 2* ladoMayor
  ESCRIBIR(”El perímetro del rectángulo de
            lados”+ ladoMenor+” y ”+ladoMayor+
            ” es”+perimetro)
FIN ALGORITMO calcularPerimetro
```

```
<?php
  /* variables: REAL $ladoMenor,
                REAL $ladoMayor,
                REAL $perimetro */
  echo "Calcula el perímetro de un rectángulo";

  echo "Ingrese el valor del lado menor";
  /*leer lo que ingresa el usuario*/
  $ladoMenor = trim(fgets(STDIN));

  echo "Ingrese el valor del lado mayor"
  /*leer lo que ingresa el usuario*/
  $ladoMayor = trim(fgets(STDIN));

  $perimetro = 2 * $ladoMenor + 2 * $ladoMayor;
  echo "El perímetro de lados ".$ladoMenor.
       "es"."$perimetro"

  ?>
```

3.11. Otros ejemplos de expresiones

No sólo tendremos expresiones numéricas en un programa PHP. Por ejemplo, veamos el siguiente programa para saludar amigos:

```
ALGORITMO saludo() RETORNA ⌀
  (* alg. que saluda a un amigo  *)
  TEXTO $alguien
  LEER($alguien)
  ESCRIBIR(" Hola " + $alguien + "!")
  ESCRIBIR(" Estoy programando en PHP.")
FIN ALGORITMO saludo
```

```
<?php
 /* Alg. que saluda a un amigo
 String $alguien */

 echo " Ingrese su nombre : " ;
 $alguien = trim ( fgets ( STDIN ));
 echo " Hola " . $alguien . "!" ;
 echo " Estoy programando en PHP." ;

? >
```

La variable $alguien queda asociada a un valor ingresado por el usuario, en este caso el usuario ingresará una cadena de caracteres (letras, dígitos, símbolos, etc.), Por ejemplo “Ana”.

Utilicemos el programa anterior y fijemos el nombre de la persona para ejemplificar las reglas de forman expresiones:

```
ALGORITMO saludo() RETORNA ⌀
  (* alg. que saluda a Ana  *)
  TEXTO $alguien
  $alguien <- "Ana"
  ESCRIBIR(" Hola " + $alguien + "!")
  ESCRIBIR(" Estoy programando en PHP.")
FIN ALGORITMO saludo
```

```
<?php
  /* Saludar a Ana */

  $alguien = "Ana";
  echo " Hola " . $alguien . "!" ;
  echo " Estoy programando en PHP." ;

?>
```

```
##  Hola Ana! Estoy programando en PHP.
```

(Observación: PHP usa también una notación con comillas simples para referirse a las cadenas de caracteres, y hablar de ’Ana’.) Como en la sección anterior, podemos enumerar las reglas que forman expresiones con caracteres:

- Un valor literal también es una expresión. Por ejemplo el resultado de la expresión “Ana” es precisamente “Ana”.
- Una variable es una expresión, y el valor que produce es el que tiene almacenado (si $alguien almacena el valor “Ana”, entonces el resultado de la expresión $alguien es la cadena “Ana”).
- Usamos operaciones para combinar expresiones y construir expresiones más complejas, pero atención con qué operaciones están permitidas sobre cadenas:
- El signo “.” representa la concatenación de cadenas: La expresión “Hola”.$alguien.“!”, es una expresión cuyo resultado es “Hola Ana!”.

3.12. Instrucciones

Las instrucciones son las órdenes que entiende PHP. Ya hemos usado varias instrucciones:

- hemos mostrado valores por pantalla mediante la instrucción **echo**,
- hemos Leido valores de mediante la instrucción **$variable = trim(fgets(STDIN))**,
- hemos asociado valores a una variables mediante la instrucción de asignación.