UNIDAD 3: INTRODUCCIÓN A PHP

Cátedra de Introducción a la Programación
Facultad de Informática
Universidad Nacional del Comahue

---

1. INTRODUCCIÓN

En esta unidad empezaremos a resolver problemas sencillos, y a programarlos en **PHP**.

1.1. 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**. Éste es el punto en el cual se describe qué debe hacer el programa, sin importar el 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**. Éste 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.

1.2. 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 útil?

—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 útil de los datos de entrada? ¿Pueden pensar qué información es útil 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?

1.3. 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

ATENCIÓN

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*)
    FLOAT 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
<?php
/*Suma tres longitudes en millas, pies y pulgadas convirtiéndolas en metros */
/*float $millas,$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.

ATENCIÓN

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).

Palabras reservadas de PHP				
__halt_compiler()	abstract	and	array()	as
break	callable (a partir de PHP 5.4)	case	catch	class
clone	const	continue	declare	default
die()	do	echo	else	elseif
empty()	enddeclare	endfor	endforeach	endif
endswitch	endwhile	eval()	exit()	extends
final	finally (a partir de PHP 5.5)	for	foreach	function
global	goto (a partir de PHP 5.3)	if	implements	include
include_once	instanceof	insteadof (a partir de PHP 5.4)	interface	isset()
list()	namespace (a partir de PHP 5.3)	new	or	print
private	protected	public	require	require_once
return	static	switch	throw	trait (a partir de PHP 5.4)
try	unset()	use	var	while
xor	yield (a partir de PHP 5.5)			

Fig. 1. Palabras Reservadas en PHP. http://php.net/manual/es/reserved.keywords.php

1.4. Piezas de un programa PHP

Para poder empezar a programar en PHP es necesario conocer los elementos que constituyen un programa en dicho lenguaje y las reglas para construirlos.

SABIAS QUE...

Cuando empezamos a hablar en un idioma extranjero es posible que nos entiendan pese a que cometamos errores. No sucede lo mismo con los lenguajes de programación: la computadora no nos entenderá si nos desviamos un poco de alguna de las reglas.

1.4.1. Nombres

Ya hemos visto que se usan nombres para denominar a los programas (progConvertirSistemaMetrico) y para denominar a las funciones dentro de un módulo (convertirSistemaMetrico). Cuando queremos dar nombres a valores usamos variables (*millas, pies, pulgadas, metros*). Todos esos nombres se llaman identificadores y PHP tiene reglas sobre qué es un identificador válido y qué no lo es. Un identificador comienza con el simbolo "$" y luego sigue con una secuencia de letras, números y guiones bajos. Los espacios no están permitidos dentro de los identificadores. Los siguientes son todos identificadores válidos de **PHP**:

—$hola
—$hola12t
—$_hola
—$Hola

PHP distingue mayúsculas de minúsculas, así que Hola es un identificador y hola es otro identificador. Los siguientes son todos identificadores inválidos de **PHP**:

—$hola a12t
—$8hola
—$hola %
—$Hola*9

PHP reserva palabras para describir estructura del programa, y no permite que se usen 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 esta lista 1, a la que llamaremos de **palabras reservadas**.

Iniciar un programa PHP

Para indicar que comenzamos a escribir instrucciones en lenguaje PHP debemos utilizar el tag *<?php*. Cuando terminamos de escribir código PHP cerramos utilizando *?>*, aunque si el archivo con el código fuente sólo contienen código PHP, este tag no es necesario.

1.4.2. Convención para nombrar variables

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).

Para nombrar variables utilizaremos la notación "lowerCamelCase".

CamelCase es un estilo de escritura que se aplica a frases o palabras compuestas. El nombre se debe a que las mayúsculas a lo largo de una palabra en CamelCase se asemejan a las jorobas de un camello. El nombre CamelCase se podría traducir como Mayúsculas/Minúsculas Camello.

Existen dos tipos de CamelCase:

—**UpperCamelCase**, cuando la primera letra de cada una de las palabras es mayúscula. Ejemplo: EjemploDeUpperCamelCase.

—**lowerCamelCase**, igual que la anterior con la excepción de que la primera letra es minúscula. Ejemplo: ejemploDeLowerCamelCase.

PHP es sensible!!!

PHP es case sensitive, es decir, sensible a las mayúsculas y minúsculas. Por ejemplo las variables $var, $Var, $VAR son variables distintas.
Hay que ser cuidadoso al escribir los nombres de variables.

1.4.3. Tipos de Datos

Un Tipo de Dato determina los valores que puede tomar una variable y las operaciones que pueden realizarse con ella. Todos los lenguajes de Programación definen sus tipos de datos.

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

1.4.4. 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 número 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 números 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.

1.5. No sólo de números viven los programas

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

```
1 <?php
2     /*Saludar a un amigo*/
3
4   echo "Ingrese␣su␣nombre:␣" ;
5   $alguien = trim(fgets(STDIN));
6   echo "Hola␣" . $alguien . "!" ;
7   echo "Estoy␣programando␣en␣PHP." ;
8
9 ?>
```

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

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

```
1  <?php
2      /*Saludar a Ana*/
3
4    $alguien = "Ana"
5    echo "Hola␣" . $alguien . "!" ;
6    echo "Estoy␣programando␣en␣PHP." ;
7
8  ?>
```

(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 qué 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!".

1.6. 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.