Capítulo 2: Introducción al lenguaje.

PHP es un lenguaje no posicional, por lo que no importa la columna en la cual se comience a escribir el código. Tampoco influye sobre el código la cantidad de saltos de línea (enter) que se coloquen, ni la cantidad de espacios.

La forma en la que se separan las distintas sentencias es mediante la utilización de ";". En PHP cada sentencia debe finalizar con ";".

Se puede escribir más de una sentencia en la misma línea siempre y cuando las mismas se encuentren separadas con ";".

Comentarios:

En PHP hay 3 formas distintas de incluir comentarios:

```
/* Al estilo de C
   en donde el comentario empieza
   y termina delimitado por barra asterisco y asterisco barra
*/

O bien usando
// Comentario

O por último
# Comentario
```

En las dos últimas variantes el comentario empieza en donde se encuentra el "//" o el "#" y termina cuando termina la línea.

Tipos de Datos:

PHP soporta los siguientes tipos de datos:

- Enteros
- Vectores
- Binarios de punto flotante
- Strings
- Objetos

En general el tipo de dato de una variable no es decidido por el programador sino que lo decide el lenguaje en tiempo de ejecución, la instrucción settype puede usarse para forzar el tipo de dato de una variable en los raros casos en que esto sea necesario. Todas las variables en php se denotan utilizando el signo '$' precediendo al nombre de la variable.

Enteros:

```
$a = 1234; # número decimal
$a = -123; # número negativo
$a = 0123; # número octal  (83 en decimal)
$a = 0x12; #número en hexadecimal (18 decimal)
```

Flotantes:

Los números de punto flotante pueden notarse de la siguiente manera:

```
$a = 1.234;
$a = 1.2e3;
```

Strings:

En PHP los strings tienen un manejo similar al utilizado en “C” o “C++”, están predefinidos los siguientes caracteres especiales:

\n	Nueva línea
\r	Salto de carro (carring return)
\t	Tabulación
\\	Barra invertida
\$	Signo pesos
\”	Comillas doble

Un string puede inicializarse usando comillas dobles o comillas simples. Cuando se utilizan comillas dobles el interprete de php parsea previamente el string, es decir que reemplaza los nombres de variables que encuentra en el string por el contenido de la variable. Cuando se usan comillas simples el string se imprime tal y como figura sin ser parseado.

Ej:

```
$x=”Juan”;
$s=”Hola $x”;
$t=’Hola $x’
```

$s vale “Hola Juan” y $t vale “Hola $x”.

Otra forma de inicializar un string es usando un string multilinea de la siguiente manera:

```
$str=<<<EOD
Este es un ejemplo de un string
que ocupa varias líneas y se puede
definir así
EOD;
```

Se pueden concatenar strings usando el operador “.” de la siguiente manera:

```
$x=”hola”;
$y=”mundo”;
$s=$x.$y; #$s es igual a “holamundo”.
$s=$x.” “.$y; # Aquí $s vale “hola mundo”
```

Vectores:

Los vectores en php actúan tanto como vectores tradicionales (indexados por número) así también como vectores asociativos (indexados por clave).

Los vectores pueden crearse usando las instrucciones “list” o “array” o bien inicializando en forma explícita cada elemento del vector.

```
$a[0]=”hola”
$a[1]=”mundo;
$a[“clave”]=”valor”;
```

Utilizando la notación especial $v[]; se pueden agregar elementos a un vector.

```
$a[0]=”nada”;
$a[1]=”hola”;
$a[]=”mundo”;  #Asigna a $a[2] el valor “mundo”.
```

Existen funciones especiales para ordenar vectores, contar la cantidad de elementos de los mismos, recorrerlos, etc. (Ver el capítulo sobre vectores)

Matrices:

La definición, inicialización y uso de matrices en PHP es sencilla. Se puede pensar una matriz en PHP como un vector de vectores, por lo que se puede utilizar la misma lógica que en los primeros.

```
$a[0][1]=”Hola”;
$a[0][“clave”]=”una cosa”;
$a[“clave1”][“clave2”][0][1]=”otra cosa”;
etc…
```

Para incluir el valor de un elemento de un vector en un string se deben usar llaves para indicar el alcance del nombre de la variable a reemplazar:

```
Echo “Esta es una prueba {$a[0][1]}”;
```

Una forma útil de inicializar un vector asociativo es usando la siguiente notación:

```
$a=array(
  “color” => “rojo”,
  “edad” => 23,
  “nombre” => “juan”
);
```

Para crear una matriz se pueden anidar las declaraciones de tipo array.

```
$a = array(
       "apple"  => array(
            "color"  => "red",
            "taste"  => "sweet",
            "shape"  => "round"
       ),
       "orange"  => array(
           "color"  => "orange",
           "taste"  => "tart",
            "shape"  => "round"
       ),
       "banana"  => array(
            "color"  => "yellow",
            "taste"  => "paste-y",
            "shape"  => "banana-shaped"
       )
  );
```

Constantes:

Para definir una constante se utiliza la instrucción “define” de la forma:

define(“PI”,3.14151692);

Luego las constantes pueden usarse como variables tradicionales ($PI) con la salvedad de que no se les puede asignar un valor.

Operadores:

Los operadores aritméticos en PHP también se asemejan al C:

```
$a + $b;        //suma
$a - $b;        //resta
$a++;           //pos-incremento, esta sentencia devuelve el valor de $a y lo incrementa en 1.
++$a;           //pre-incremento, incrementa en 1 el valor de $a y devuelve el valor incrementado.
$a--;           //pos-decremento
--$a;           //pre-decremento
$a * $b;        //multiplicación
$a / $b;        //división
$a % $b;        //módulo
```

Asignación:

La asignación se resuelve con el signo igual (“=”).

```
$a=5;           //Asigna 5 a $a
$a=$b;          //Asigna el valor de $b a $a
$b=($c=6);      //Asigna 6 a $c y a $b
```

Y pueden combinarse asignación y operadores de la forma:

```
$a+=5;          //Suma y asigna 5 a $a
$x.=”hola”;     //Concatena $x con “hola”
```

Operaciones con bits:

```
$a & $b;        //$a AND $b
$a | $b;        //$a OR $b
$a ^ $b;        //$a XOR $b
~$a;            //NOT $a
$a << $b;       //Shift hacia la izquierda $b posiciones
$a >> $b;       //Shift hacia la derecha $b posiciones
```

Comparaciones:

```
$a = = $b;      //true si $a igual a $b
$a = = = $b;    //true si $a igual a $b y además son del mismo tipo
$a >= $b;       //mayor o igual
$a <= $b;       //menor o igual
$a != $b;       //true si a y b son distintos
```

Operador @

Cuando se antepone @ a una expresión se suprimen los errores que la expresión pudiera generar.

Ej:

@($c=$a/$b);
Operador de ejecución:

PHP soporta el uso de “backticks” (comillas invertidas) para ejecutar un comando desde el shell y devolver el resultado de la ejecución del comando en una variable:

$result=`ls –l`;

Operadores lógicos:

```
$a && $b;       //True si $a es true y $b es true
$a || $b;       //True si $a es true o $b es true
$a xor $b;      //Or exclusivo
!$a;            //True si $a es falso (NOT)
```

Estructuras de Control:

If:

```
if (expresión) sentencia;

if (expresión) {sentencias;}

if (expresión) {
  sentencias;
} else {
  sentencias;
}

if (expresión) {
  sentencias;
} elseif (expresión) {
  sentencias;
} else (expresión) {
  sentencias;
}
```

While:

```
while (expresión) {
 sentencias;
}

do {
 sentencias;
} while(expresión)
```

For:

```
for (expr1,expr2,expr3) {
 sentencias;
}
```

La primera expresión cumple la función de inicializar las variables de control del FOR. Esta expresión se cumple incondicionalmente, más allá de que se entre dentro del ciclo o no.
La expresión 2 se evalúa siempre que se este por ingresar al ciclo del FOR, aún cuando se ingresa al for por primera vez.
La tercera expresión se ejecuta cada vez que se termina el ciclo. Por lo general se utiliza esta expresión para indicar el incremento de alguna variable que se este utilizando para el FOR. La ejecución de esta expresión es también incondicional y es la que se ejecuta inmediatamente antes de evaluarse la expresión 2.

Ejemplo:

```
for ($i=0;$i<5;$i++) {
 print("$i");
}
```

Foreach:
Para realizar ciclos con la cantidad de ocurrencias en un vector se utiliza el comando foreach:

```
foreach ($vector as $variable) {
 sentencias;
}

foreach ($vector as $clave => $valor) {
 sentencias;
}
```

Por cada iteración cada elemento de $vector es asignado a $variable.
El segundo caso es aplicable para recorrer vectores asociativos.

Break:

Break permite salir del ciclo actual “for” , “while” o “switch”

Ejemplo:

```
for ($i=0;$i<6;$i++) {
 If($i==$b) break;   //Sale del ciclo si $i es igual a $b
}
```

Switch:

El switch permite ejecutar un grupo de sentencias de acuerdo al valor de una variable:

```
switch ($variable) {
 case valor:
    sentencias;
    break;
  case valor2:
    sentencias;
    break;
  default:
    sentencias;
    break;
}
```

Cuando el valor de la variable ($variable) coincide con el valor de algún “case”, se ejecutan las sentencias que se encuentran a continuación.

En este caso se utiliza la sentencia “break” en forma prácticamente obligatoria, porque en caso de no existir esta sentencia se seguiría ejecutando linealmente todas las sentencias continuas, es decir las sentencias de los demás “cases” inferiores.

Por último, la opción “default” se utiliza generalmente para cuando el valor de la variable no coincide con ningún “case”. Estas sentencias se ejecutan siempre, salvo en el caso de que se ejecute antes un “break”.

Funciones:

Definir subrutinas (funciones) en php es sencillo:

```
function prueba($a,$b) {
  $r=$a + $b;
  return $r;
}
```

Para invocar a la función basta con hacer $x=prueba(4,6);
Los parámetros que recibe la función pueden ser enteros, flotantes, strings, vectores u objetos es decir cualquiera de los tipos de datos soportado por PHP.
El valor devuelto por la función también puede ser cualquier tipo de datos de php.

Parámetros default:

Es posible asignar un valor default a los parámetros que recibe una función de forma tal que cuando se invoca la función si se ignora el parámetro el mismo es asignado al default.

```
function prueba($a=2,$b=3,$c=5) {
  //código
}
```

Si se llama prueba(4) // Entonces $a=4, $b=3 y $c=5

Se puede saber cuantos parámetros recibió una función usando func_num_args() y se puede obtener el iesimo parámetro de una función con func_get_arg(número_de_parámetro);

Finalmente los nombres de funciones pueden guardarse en variables e invocarse las mismas usando el nombre guardado en una variable.

```
Ej: $nombre=sumar;
$nombre(4,5);      //Llama a la función sumar
```

Esto permite guardar nombres de funciones en tablas, archivos, vectores etc lo cual da lugar a ciertas maniobras interesantes que no parece demasiado conveniente enumerar en este capítulo.