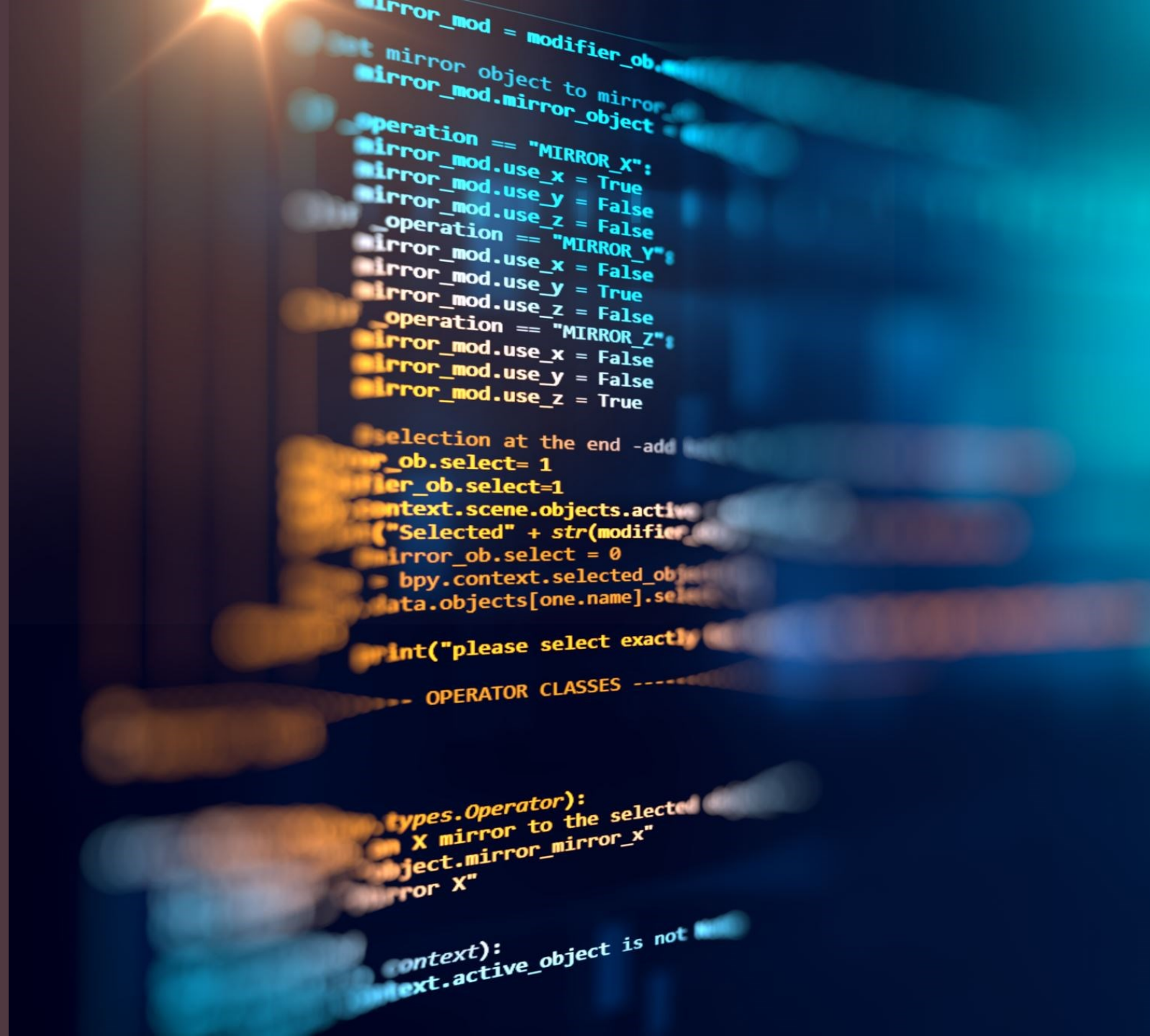
Variables, Constantes, Funciones y Operadores en Php

https://www.php.net/manual/es/



Variables

Nombres de las variables en php

En PHP todos los nombres de variable tienen que empezar por el símbolo **$**

Los nombres de las variables han de **llevar una letra inmediatamente después del símbolo $ o bien _ aunque no se recomienda su uso puesto que PHP lo utiliza para definir las variables superglobales:**

$pepe1 es un nombre válido, pero **$1pepe** *no es un nombre válido*

Para PHP las letras ***mayúsculas* y las *minúsculas* son distintas:**

la variable $pepe es **distinta** de $Pepe

Tipos de variables.

En PHP no es necesario definir el **tipo de variable**, por lo tanto, **una misma variable** puede contener una *cadena de caracteres* en un momento del proceso y, posteriormente, un *valor numérico*, susceptible de ser operado matemáticamente.

NOTA: se recomienda usar una variable siempre con el mismo tipo de dato.

Variables

Definición de variables.

PHP no requiere una definición previa de las variables. Se definen en el momento en que son necesarias y para ello basta que se les asigne un valor.

La sintaxis es esta:

$variable=valor;

El ***valor*** puede ser una *cadena* (texto o texto y números que no requieren ser operados matemáticamente) o sólo un *número*.
En el caso de las cadenas habría que escribir el valor **entre comillas**.

$variable="Esto es una cadena";

Variables

Aclaraciones

PHP no requiere (“ni soporta”) la definición explicita de tipos en la declaración de variables.

Dependiendo del valor que se le asigne, a la variable se le aplica un tipo de datos, que puede cambiar el tipo si cambia su contenido.

Esto es, el tipo de la variable se decide en función del contexto en que se emplee.

```
// Al asignarle el valor 7, la variable es de tipo entero
$mi_variable = 7;
// Si le cambiamos el contenido $mi_variable = "siete";
// La variable cambia de tipo

// En este caso pasa a ser de tipo cadena
// Al asignarle el valor 7, la variable es de tipo entero
$mi_variable = 7; // Si le cambiamos el contenido
$mi_variable = "siete";// La variable cambia de tipo
// En este caso pasa a ser de tipo cadena
```

Variables

Tipos de datos simples en PHP son:

- **booleano** (boolean). Sus posibles valores son true y false. Además, cualquier número entero se considera como true, salvo el 0 que es false.

- **entero** (integer). Cualquier número sin decimales. Se pueden representar en formato decimal, octal (comenzando por un 0), o hexadecimal (comenzando por 0x). Tanto echo como print nos mostrarán el dato en notación decimal.

- **real** (float). Cualquier número con decimales. Se pueden representar también en notación científica.

- **cadena** (string). Conjuntos de caracteres delimitados por comillas simples o dobles.

- **NULL** Es un tipo de datos especial, que se usa para indicar que la variable no tiene valor. Nulo en PHP se considera un tipo de datos.
Esto significa, realmente, "ningún valor". Es decir, no representa un cero, ni una cadena vacía, ni un false. Es el valor de una variable cuando aún no se le ha asignado nada.

Variables

Tipos de datos compuestos en PHP son:

- **array:** colección de variables guardadas bajo un mismo nombre, diferenciadas entre sí por un índice.
  En php hay dos tipos:
  - arrays indexados
  - arrays asociativos

- **object.** Instancia de una clase.

Tipos de datos especiales:

- **resource:** Un valor tipo resource es una variable especial, que contiene una referencia a un recurso externo, como un fichero o una base de datos. Los recursos son creados y usados por funciones especiales

Variables

Función gettype

gettype: Obtener el tipo de una variable

(PHP 4, PHP 5, PHP 7, PHP 8)

gettype(mixed $var): string

Devuelve el tipo de la variable PHP $var.
Para la comprobación de tipos, utilice las funciones is_*.

Parámetros
Var la variable de la cual queremos comprobar su tipo.

Variables

Función gettype

Valores devueltos
Los valores posibles para la cadena devuelta son:
"**boolean**"
"**integer**"
" float " o "double" (por razones históricas "double" es devuelto en caso de que un valor sea de tipo float)
"**string**"
"**array**"
"**object**"
"**resource**"
"**NULL**"
"**unknown type**"

Operadores aritméticos

Operadores	Representación
Suma	$x + $y
Resta	$x - $y
Multiplicación	$x * $y
División	$x / $y
Módulo	$x % $y (resto de la división)
Exponenciación	$x ** $y
Negación	-$x

División devuelve un (int) si $x y $y son divisibles, o (float) si no lo son.

En Módulo se eliminarían primero las partes decimales transformándose en (int) en caso de ser (float) y luego se haría la operación. El signo (+ o -) del resultado dependerá del dividendo, por ejemplo: -5 % 3 mostraría -2.

Operadores aritméticos

Ejemplo	Nombre	Resultado
$a == $b	Igual	**TRUE** si *$a* es igual a *$b* después de la manipulación de tipos.
$a === $b	Idéntico	**TRUE** si *$a* es igual a *$b*, y son del mismo tipo.
$a != $b	Diferente	**TRUE** si *$a* no es igual a *$b* después de la manipulación de tipos.
$a <> $b	Diferente	**TRUE** si *$a* no es igual a *$b* después de la manipulación de tipos.
$a !== $b	No idéntico	**TRUE** si *$a* no es igual a *$b*, o si no son del mismo tipo.
$a < $b	Menor que	**TRUE** si *$a* es estrictamente menor que *$b*.
$a > $b	Mayor que	**TRUE** si *$a* es estrictamente mayor que *$b*.
$a <= $b	Menor o igual que	**TRUE** si *$a* es menor o igual que *$b*.
$a >= $b	Mayor o igual que	**TRUE** si *$a* es mayor o igual que *$b*.
$a <=> $b	Nave espacial	Un integer menor que, igual a, o mayor que cero cuando *$a* es respectivamente menor que, igual a, o mayor que *$b*. Disponible a partir de PHP 7.

Operadores aritméticos

1. Si se compara un número con un string o la comparación implica strings numéricos, entonces cada string es convertido en un número y la comparación se realiza numéricamente.

1. PHP 7 introduce un nuevo tipo de operador, que se puede utilizar para comparar expresiones llamado **nave espacial** cuyo resultado es:

   $a <=> $b evalúa a:

0	si	$a == $b
-1	si	$a < $b
1	Si	$a > $b

Bucle for. Sintaxis

```
for (inicialización; condición; incremento)

for (expr1; expr2; expr3)
{
   instrucciones
}
```

Ejemplo:

```
<?php
   for($i=1; $i<=15; $i++){
      echo $i;
      echo "<br>";
   }
?>
```

Ejercicio. Tabla de multiplicar

Operando1	Operador	Operando2	Resultado
1	x	0	0
1	x	1	1
1	x	2	2
1	x	10	10

Operadores

Operadores lógicos

Ejemplo	Nombre	Resultado
$a and $b	And (y)	**TRUE** si tanto *$a* como *$b* son **TRUE**.
$a or $b	Or (o inclusivo)	**TRUE** si cualquiera de *$a* o *$b* es **TRUE**.
$a xor $b	Xor (o exclusivo)	**TRUE** si *$a* o *$b* es **TRUE**, pero no ambos.
! $a	Not (no)	**TRUE** si *$a* no es **TRUE**.
$a && $b	And (y)	**TRUE** si tanto *$a* como *$b* son **TRUE**.
$a || $b	Or (o inclusivo)	**TRUE** si cualquiera de *$a* o *$b* es **TRUE**.

La razón para tener las dos variaciones diferentes de los operadores "and" y "or" es porque operan con precedencias diferentes.

Operadores asignación

Operador	Ejemplo	Equivalencia
=	$a=$b;	$a toma el valor de $b
+=	$a += $b;	$a = $a + $b
-=	$a -= $b	$a = $a - $b
*=	$a *= $b;	$a = $a * $b;
/=	$a /= $b;	$a = $a / $b;
%=	$a %= $b;	$a = $a % $b;
.=	$a .= $b;	$a = $a . $b;

Operadores de incremento y decremento

Los operadores de incremento y decremento sólo afectan a números y strings, sin afectar a arrays, objects o resources.

Decrementar un valor NULL no tiene efecto, pero si se incrementa se obtiene 1. Incrementar o decrementar booleanos no tiene efecto.

Operador	Efecto
++$x	Incrementa $x en 1 y devuelve $x
$x++	Retorna $x y luego incrementa $x en 1
--$x	Decrementa $x en 1 y devuelve $x
$x--	Retorna $x y luego decrementa $x en 1

Operadores de incremento y decremento

```php
<?php
    $x = 4;
    echo "Esto es 4: " . $x++ . "<br>";
    echo "Y esto es 5: " . $x . "<br>";

    $x = 4;
    echo "Esto es 5: " . ++$x . "<br>";
    echo "Y esto es 5: " . $x . "<br>";

    $x = 4;
    echo "Esto es 4: " . $x-- . "<br>";
    echo "Y esto es 3: " . $x . "<br>";
?>
```

Operadores de incremento y decremento

```php
<?php
    //Al trabajar sobre caracteres, de Z pasa a AA. Por ejemplo:
    $x = 'Z';
    echo ++$x; // Devolverá AA
    echo ++$x; // Devolverá AB

    $x = 'A9';
    echo ++$x; // Devolverá B0
    echo ++$x; // Devolverá B1

    $x = 'A09';
    echo ++$x; // Devolverá A10
    echo ++$x; // Devolverá A11
?>
```

Precedencia de operadores

- La precedencia de un operador indica cómo se evalúan dos expresiones juntas.
  Por ejemplo, en la expresión *1 + 5 * 3* , la respuesta es *16* y no *18*
  porque el operador de multiplicación ("*") tiene una precedencia mayor que el operador de adición ("+").

- Los paréntesis pueden ser usados para forzar la precedencia, si es necesario.

  Por ejemplo: *(1 + 5) * 3* se evalúa como *18*.

- Cuando los operadores tienen igual precedencia su asociatividad decide cómo se agrupan.

  Por ejemplo:
  "-" tiene asociatividad a izquierda, así *1 - 2 - 3* se agrupa como *(1 - 2) - 3* y se evalúa a *-4*.
  "=", por otra parte, tiene asociatividad a derecha, así *$a = $b = $c* se agrupa como *$a = ($b = $c)*.

- Los operadores de igual precedencia que no son asociativos no pueden usarse unos junto a otros,
  Por ejemplo:
  1 < 2 > 1 es ilegal en PHP, puesto que no son asociativos
  La expresión *1 <= 1 == 1*, por otro lado, es legal, ya que el operador == tiene menos precedencia que el operador <=.

El uso de paréntesis, incluso cuando no es estrictamente necesario, a menudo puede aumentar la legibilidad del código haciendo grupos explícitamente en lugar de confiar en la precedencia y asociatividad implícitas del operador.

La siguiente tabla enumera los operadores en orden de precedencia, con los de más alta precedencia al inicio. Los operadores en la misma línea tienen igual precedencia, en cuyo caso la asociatividad decide el agrupamiento.

Operadores

Precedencia de operadores

Asociatividad	Operadores	Información adicional
izquierda	*[*	array()
derecha	**	aritmética
derecha	++ -- *(int) (float) (string) (array) (object) (bool)* @	tipos e incremento/decremento
derecha	*!*	lógico
izquierda	* / %	aritmética
izquierda	+ - .	aritmética y string
no asociativo	< <= > >=	comparación
no asociativo	== != === !== <> <=>	comparación
izquierda	&&	lógico
izquierda	||	lógico
derecha	??	comparación
izquierda	? :	ternario
derecha	= += -= *= **= /= .= %=	asignación
izquierda	*and*	lógico
izquierda	*xor*	lógico
izquierda	*or*	lógico

Constantes

Sintaxis

Se puede definir una constante usando la función define() o con la palabra reservada const

define()
Define una constante con nombre en tiempo de ejecución

Descripción

```
define ( string $name , mixed $value, bool $case_insensitive = false  ) : bool
```

mixed indica que un parámetro puede aceptar múltiples tipos, pero no necesariamente todos.

Constantes

Sintaxis

Parámetros de la función define

name
El nombre de la constante.

value
El valor de la constante. En PHP 5, value debe ser un valor escalar (integer, float, string, boolean, o NULL). En PHP 7, también se aceptan valores de tipo array.

case_insensitive
Si está establecido a TRUE, la constante será definida insensible a mayúsculas y minúsculas. El comportamiento predeterminado es sensible a mayúsculas y minúsculas; esto es, *CONSTANTE* y *Constante* representan valores diferentes.

Nota:
Las constantes insensibles a mayúsculas yminúsculas se almacenan en minúsculas.

Valores devueltos

Devuelve TRUE en caso de éxito o FALSE en caso de error.

Constantes

Ejemplo

```php
<?php
    define ("PI", 3.1415926);
    define ("BR", "<br>");
    define ("LIBRO", "Libro de PHP 7");
    echo PI;
    echo BR;
    echo LIBRO;
?>
```

Constantes

Palabra reservada const

- A partir de PHP 5.3.0 podemos definir una constante con la palabra reservada const fuera de la definición de una clase. Antes solo se podía utilizar dentro de una clase.

- Antes de PHP 5.6, al emplear la palabra reservada const, solamente los datos escalares (boolean, integer, float y string) podían estar contenidos en constante.

- Desde PHP 5.6 en adelante, es posible definir una constante como una expresión escalar, y también es posible definir un array constante.

Constantes

Ejemplo

```php
<?php
// Funciona a partir de PHP 5.3.0
const CONSTANTE = 'Hola Mundo';

echo CONSTANTE;

// Funciona a partir de PHP 5.6.0
const OTRA_CONSTANTE = CONSTANTE.'; Adiós Mundo';
echo OTRA_CONSTANTE;

const UNO = 1;
const DOS = UNO * 2;
echo DOS;
?>
```

Constantes

Estas son las diferencias entre constantes y variables:

- Las constantes no llevan el signo dólar ($) como prefijo.

-Antes de PHP 5.3, las constantes solo podían ser definidas usando la función define(), y no por simple asignación.

-Las constantes pueden ser definidas y accedidas desde cualquier sitio del script. Es decir**, se verán dentro de las funciones**.

-Las constantes no pueden ser redefinidas o eliminadas una vez se han definido.

Constantes

Estas son las diferencias entre constantes y variables:

La diferencia crucial entre las dos formas de declarar constantes en PHP.

A diferencia de definir constantes usando **define()**, las constantes definidas con la palabra clave **const** deben declararse en el nivel superior del entorno de la aplicación porque se definen en tiempo de compilación.

Es decir, las constantes **definidas con const no pueden declararse** dentro de funciones, bucles, sentencias **if** o bloques **try/ catch**.

Estructuras de control

Condicionales y Repetitivas o bucles

Condicionales: **if / elseif / else**

if / elseif / else: permite definir una expresión para ejecutar o no la sentencia o conjunto de sentencias siguientes:

```php
<?php
    if ($a < $b)
        print "a es menor que b";
    elseif ($a > $b)
        print "a es mayor que b";
    else
        print "a es igual a b";
?>
```

Estructuras de control

Existen dos tipos: Condicionales y repetitivas o bucles

Condicionales: **switch-case**

switch: similar a enlazar varias sentencias if comparando una misma variable con diferentes valores

```
<?php
        switch ($a) {
            case 0:
                print "a vale 0";
                break;
            case 1:
                print "a vale 1";
                break;
            default:
                print "a no vale 0 ni 1";
        }
?>
```

Estructuras de control

Condicionales y repetitivas o bucles

Repetitiva: while

while: define un bucle que se ejecuta mientras se cumpla una expresión. La expresión se evalúa antes de comenzar cada ejecución del bucle.

```php
<?php
        $a = 1;
        while ($a < 8) {
            $a += 3;
        }
        print $a; // el valor obtenido es 10
?>
```

Estructuras de control

Condicionales y repetitivas o bucles

Repetitiva: do...while

do..while: similar al bucle while, pero la expresión se evalúa al final, con lo cual se asegura que la sentencia/as del bucle se ejecutan al menos una vez.

```php
<?php
        $a = 1;
        do{
            $a -=3;
        }while ($a > 10);
        print $a; // el valor obtenido es -2
?>
```

Estructuras de control

Existen dos tipos: Condicionales y repetitivas o bucles

Repetitiva: for

for: compuesto por tres expresiones:

for (expr1; expr2; expr3)

sentencia o conjunto de sentencias;

expr1: se ejecuta una vez al comienzo del bucle.
expr2: se evalúa para saber si se debe ejecutar o no la/as sentencia/as
expr3: se ejecuta tras ejecutar todas las sentencias del bucle

```php
<?php
    for ($a = 5; $a<10; $a+=3) {
        print $a; // Se muestran los valores 5 y 8
        print "<br />";
    }
?>
```

Ámbito de las variables en PHP

En PHP puedes utilizar variables en cualquier lugar de un programa. Si esa variable aún no existe, la primera vez que se utiliza se reserva espacio para ella.

En ese momento, dependiendo del lugar del código en que aparezca, se decide desde que partes del programa se podrá utilizar esa variable.

A esto se le llama **visibilidad de la variable**.

Ámbito de las variables en PHP

Si la variable aparece por primera vez dentro de una función, se dice que esa variable es local a la función.

Si aparece una asignación fuera de la función, será considerada una variable distinta.

```php
<?php
    $a = 1;
    function prueba()
    {
        $b = $a;
        // Dentro de la función no se tiene acceso a la variable $a
        // Por tanto, la variable $a usada en la asignación anterior es una
variable nueva
        // que no tiene valor asignado (su valor es null)
    }
?>
```

Ámbito de las variables en PHP

Si en la función anterior quisieras utilizar la variable $a externa, podrías hacerlo utilizando la palabra global. De esta forma le dices a PHP que no cree una nueva variable local, sino que utilice la ya existente.

```php
<?php
    $a = 1;
    function prueba()
    {
        global $a;
        $b = $a; //En este caso se le asigna a $b el valor 1
    }
?>
```

Ámbito de las variables en PHP

Las variables locales a una función desparecen cuando acaba la función y su valor se pierde.
Si quisieras mantener el valor de una variable local entre distintas llamadas a la función, deberás declarar la variable como estática utilizando la palabra static.

Las variables estáticas deben inicializarse en la misma sentencia en que se declaran como estáticas. De esta forma, se inicializan sólo la primera vez que se llama a la función.

```php
<?php
    function contador()
    {
        static $a = 0;
        $a++; // Cada vez que se ejecuta la función, se incrementa el valor de $a
    }
?>
```

Resumen

-En PHP, todas las variables creadas en la página, fuera de funciones, son variables globales a la página.

-Las variables creadas dentro de una función son variables locales a esa función.

-Las variables globales se pueden acceder en cualquier lugar de la página, mientras que las variables locales sólo tienen validez dentro de la función donde han sido creadas.

-De modo que una variable global la podemos acceder dentro de cualquier parte del código excepto desde una función.

-Mientras que si intentamos acceder a una variable local fuera de la función donde fue creada, nos encontraremos con que esa variable no tiene contenido alguno.
- Si queremos utilizar una variable global a la página dentro de una función, tenemos que especificarlo con la palabra **global**.

-Cualquier alteración que hagamos a las variables globales dentro de la función **permanecerá** cuando se haya salido de la función.

FUNCIONES DE USUARIO

Para crear tus propias funciones, deberás usar la palabra **function**.

```
<?php
    function precio_con_iva() {
        global $precio;
        $precio_iva = $precio * 1.18;
        print "El precio con IVA es ".$precio_iva;
    }
    $precio = 10;
    precio_con_iva();
?>
```

Para hacer una llamada a una función, basta con poner **su nombre** y unos paréntesis:

```
    precio_con_iva();
```

FUNCIONES DE USUARIO

En PHP no es necesario que definas una función antes de utilizarla.

Hay una excepción: cuando está condicionalmente definida como se muestra en el siguiente ejemplo:

```
<?php
$iva = true;
$precio = 10;
precio_con_iva(); // Da error, pues aquí aún no está definida la
función
if ($iva) {
    function precio_con_iva() {
        global $precio;
        $precio_iva = $precio * 1.18;
        print "El precio con IVA es ".$precio_iva;
    }
}
precio_con_iva(); // Aquí ya no da error
?>
```

FUNCIONES DE USUARIO: Argumentos

Malas prácticas en el ejemplo anterior:

En la función usábamos una variable global, lo cual no es una buena práctica.

Siempre es mejor utilizar argumentos o parámetros al hacer la llamada.

Además, en lugar de mostrar el resultado en pantalla o guardar el resultado en una variable global, las funciones pueden devolver un valor usando la sentencia return.

Cuando en una función se encuentra una sentencia return, termina su procesamiento y devuelve el valor que se indica.

FUNCIONES DE USUARIO: Argumentos

Podemos escribir la función anterior de la siguiente forma:

```
<?php
function precio_con_iva($precio) {
    return $precio * 1.18;
}
$precio = 10;
$precio_iva = precio_con_iva($precio);
print "El precio con IVA es ".$precio_iva
?>
```

Los argumentos se indican en la definición de la función como una lista de variables separada por comas.

No se indica el tipo de cada argumento, al igual que no se indica si la función va a devolver o no un valor (si una función no tiene una sentencia return, devuelve null al finalizar su procesamiento).

FUNCIONES DE USUARIO: Argumentos

Característica de las funciones PHP

- Al definir la función, tenemos la opción indicar valores por defecto para los argumentos.

-De forma que cuando hagamos una llamada a la función podemos **no** indicar el valor de un argumento;

-En este caso se toma el valor por defecto indicado.

FUNCIONES DE USUARIO: Argumentos


```
<?php
function precio_con_iva($precio, $iva=0.18) {
   return $precio * (1 + $iva);
}
$precio = 10;

$precio_iva = precio_con_iva($precio);//solo le paso un argumento
print "El precio con IVA es ".$precio_iva;

$precio_iva = precio_con_iva($precio,0.2);//le paso dos argumentos
print "El precio con IVA es ".$precio_iva;

?>
```

Puede haber valores por defecto definidos para varios argumentos, pero en la lista de argumentos de la función **todos ellos deben estar a la derecha de cualquier otro argumento sin valor por defecto**.

FUNCIONES DE USUARIO: paso por valor y por referencia

En los ejemplos anteriores los argumentos se pasaban **por valor:** cualquier cambio que se haga dentro de la función a los valores de los argumentos no se reflejará fuera de la función.

Si queremos que los cambios se mantengan fuera debemos definir el parámetro para que su valor se pase **por referencia**, añadiendo el símbolo **&** antes de su nombre.

```
<?php
function precio_con_iva(&$precio, $iva=0.18) {
   $precio *= (1 + $iva);
}
$precio = 10;
precio_con_iva($precio);

print "El precio con IVA es ".$precio
?>
```