INSTITUTO TECNOLÓGICO
SUPERIOR DE POZA RICA

INGENIERÍA SISTEMAS COMPUTACIONALES

Materia

Desarrollo en Internet

Nombre

Javier Alexis Mendoza Gracia

Docente

Simón García Ortiz

POZA RICA DE HGO, VER. **05 de noviembre del 2020**

Introducción

Los lenguajes de programación sirven para ejecutarse de tal forma que reilasen una tarea especifica que el programador quiera, a este grupo de instrucciones se le llama programa, un programa esta hecho para realizar una tarea mas compleja para solventa la necesidad del un usuario final, ya sea una pagina web, de escritorio, para un dispositivo móvil o para un back-end que es la parte lógica de una web.

PHP Básico

PHP, acrónimo de " *Hypertext Preprocessor* ", es un lenguaje interpretado de alto nivel incrustado en páginas HTML. La mayoría de su sintaxis es similar a C, Java y Perl. La meta de este lenguaje es permitir escribir a los creadores de páginas web, páginas dinámicas de una manera rápida y fácil.

Ejemplo. Una página básica

```
<html>
<head>
<title>Ejemplo PHP</title>
</head>
<body>
    <?php echo "Este es un ejemplo con PHP!"; ?>
</body>
</html>
```

Usar PHP implica no tener que escribir un programa con muchos comandos para crear una salida en HTML, sino escribir el código HTML con cierta parte de código en PHP incrustado en el mismo archivo que producirá la salida. El código PHP se incluye entre etiquetas especiales de comienzo y final <?php .... ?> que nos permiten entrar y salir del modo PHP.

Lo que distingue a PHP de la tecnología JavaScript, la cual se ejecuta en la máquina cliente, es que el código PHP es ejecutado en el servidor. Si tuviésemos un script similar al del ejemplo en nuestro servidor, el cliente solamente recibiría el resultado de su ejecución en dicho servidor, sin ninguna posibilidad de determinar que código generó el resultado recibido. Un servidor con estas características es configurado para que procese todos los archivos HTML que contengan scripts en PHP.

Aplicación de PHP

Al nivel más básico, PHP puede procesar información recibida de los formularios, generar páginas con contenidos dinámicos o mandar y recibir cookies.

Tal vez la característica más potente y destacable de PHP es su soporte para una gran cantidad de bases de datos. Escribir una interfaz vía web para una base de datos es una tarea simple con PHP. Las siguientes bases de datos están soportadas actualmente :

Adabas D	Ingres	Oracle (OCI7 and OCI8)
dBase	InterBase	PostgreSQL
Empress	FrontBase	Solid
FilePro	mSQL	Sybase
IBM DB2	MySQL	Velocis
Informix	ODBC	Unix dbm

Historia de PHP

PHP fue concebido en otoño de 1994 por Rasmus Lerdof. Las primeras versiones no distribuidas al público fueron usadas en un sus páginas web para mantener un control sobre quien consultaba su currículum. La primera versión disponible para el público a principios de 1995 fue conocida como "Herramientas para paginas web personales" (Personal Home Page Tools). Consistían en un analizador sintáctico muy simple que solo entendía unas cuantas macros y una serie de utilerías comunes en las páginas web de entonces; un libro de visitas, un contador y otras pequeñas cosas. El analizador sintáctico fue reescrito a mediados de 1995 y fue nombrado PHP/FI version 2. FI viene de otro programa que Rasmus había escrito y que procesaba los datos de formularios. Así que combinó las "Herramientas para paginas web personales", el "intérprete de formularios", añadió soporte para mSQL y PHP/FI vio la luz. PHP/FI creció a gran velocidad y la gente empezó a contribuir en el código.

Es difícil dar estadísticas exactas, pero se estima que a finales de 1996 PHP/FI se estaba usando al menos en 15.000 páginas web alrededor del mundo. A mediados de 1997 este número había crecido a mas de 50.000. A mediados de 1997 el desarrollo del proyecto sufrió un profundo cambio, dejó de ser un proyecto personal de Rasmus, al cual habían ayudado un grupo de usuarios y se convirtió en un proyecto de grupo mucho más organizado. El analizador sintáctico se rescribió desde el principio por Zeev Suraski y Andi Gutmans y este nuevo analizador estableció las bases para PHP versión 3.0. Gran cantidad de código de PHP/FI fue portado a PHP3 y otra gran cantidad fue escrito completamente nuevo.

Hoy en día, tanto PHP/FI, PHP3 como PHP4 se distribuyen en un gran número de productos tales como el servidor web "C2's StrongHold" y Redhat Linux. Una estimación conservativa basada en estadísticas de NetCraft (vea también Estudio de NetCraft sobre servidores web), es que más de 1.000.000 de servidores alrededor del mundo usan PHP. Para hacernos una idea, este número es mayor que el número de servidores que utilizan el "Netscape's Enterprise server" en Internet.

A la vez que todo esto está pasando, el trabajo de desarrollo de la próxima generación de PHP está en marcha. Esta versión utiliza el potente motor de scripts Zend para proporcionar altas prestaciones, así como soporta otros servidores web, además de Apache, que corren PHP como módulo nativo.

Elementos de Programación de PHP

Las instrucciones se separan igual que en C o perl - terminando cada sentencia con un punto y coma (;). La etiqueta de cierre (?>) también implica el fin de la sentencia, así lo siguiente es equivalente:

```
<?php
   echo "Esto es una prueba";
?>
<?php echo "Esto es una prueba" ?>
```

Tipos de datos

PHP soporta los siguientes tipos :

- arreglo
- números en punto flotante
- entero
- objeto
- cadena

El tipo de una variable normalmente no lo indica el programador; en su lugar, lo decide PHP en tiempo de ejecución dependiendo del contexto en el que se utilice esa variable.

Si se quisiese obligar a que una variable se convierta a un tipo concreto, se podría forzar la variable o usar la función **settype()** para ello.

En PHP el identificador de una variable siempre comienza con un signo de dólar ($).

Enteros

Los enteros se puede especificar usando una de las siguientes sintaxis:

```
$a = 1234; // número decimal
$a = -123; // un número negativo
$a = 0123; // número octal (equivalente al 83 decimal)
$a = 0x12; // número hexadecimal ( equivalente al 18 decimal )
```

Números en punto flotante

Los números en punto flotante ("double") se pueden especificar utilizando cualquiera de las siguientes sintaxis:

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

Cadenas

Las cadenas de caracteres se pueden especificar usando uno de dos tipos de delimitadores.

Si la cadena está encerrada entre dobles comillas ("), las variables que estén dentro de la cadena serán expandidas (*sujetas a ciertas limitaciones de interpretación*). Como en C y en Perl, el carácter de barra invertida (" \ ") se puede usar para especificar caracteres especiales :

Caracteres protegidos

Secuencia	**significado**
\n	Nueva línea
\r	Retorno de carro
\t	Tabulación horizontal
\\	Barra invertida
\$	Signo del dólar
\"	Comillas dobles
\[0-7]{1,3}	La secuencia de caracteres que coincida con la expresión regular es un carácter en notación octal
\x[0-9A-Fa-f]{1,2}	La secuencia de caracteres que coincida con la expresión regular es un carácter en notación hexadecimal

La segunda forma de delimitar una cadena de caracteres usa el carácter de comilla simple ('). Cuando una cadena va encerrada entre comillas simples, los únicos caracteres de escape que serán comprendidos son "\\" y "\'". Las variables dentro de una cadena delimitada por comillas simples no se expandirán dentro de una cadena.

Otra forma de delimitar cadenas es usando la sintaxis de documento incrustado (<<<). Se debe proporcionar un identificador EOD después de <<< y después la cadena.

Ejemplo de entrecomillado de cadenas con sintaxis de documento incrustado :

$str = <<<EOD Ejemplo de cadena Expandiendo múltiples líneas usando sintaxis de documento incrustado.EOD;

Nota : La sintaxis de documento incrustado fue añadida en PHP 4.

Las cadenas se pueden concatenar usando el operador ' . ' (punto). Nótese que el operador '+' (suma) no aplica para esto.

Se puede acceder a los caracteres dentro de una cadena tratándola como un arreglo de caracteres indexado numéricamente, usando una sintaxis similar a la de C. Vea el siguiente ejemplo :

Ejemplos de cadenas

```
<?php

/* Asignando una cadena. */

$str = "Esto es una cadena";

/* Añadiendo a la cadena. */

$str = $str . " con algo más de texto";

/* Otra forma de añadir, incluye un carácter de nueva línea protegido. */

$str .= " Y un carácter de nueva línea al final.\n";

/* Esta cadena terminará siendo '<p>Número: 9</p>' */

$num = 9;
$str = "<p>Número: $num</p>";

/* Esta será '<p>Número: $num</p>' */

$num = 9;
$str = '<p>Número: $num</p>';

/* Obtener el primer carácter de una cadena  */
```

```
$str = 'Esto es una prueba.';
$first = $str[0];

/* Obtener el último carácter de una cadena. */

$str = 'Esto es aún una prueba.';
$last = $str[strlen($str)-1];

?>
```

Conversión de cadenas

Cuando una cadena se evalúa como un valor numérico, el valor resultante y el tipo se determinan como sigue.

La cadena se evaluará como un doble si contiene cualquiera de los caracteres '.', 'e', o 'E'. En caso contrario, se evaluará como un entero.

El valor viene dado por la porción inicial de la cadena. Si la cadena comienza con datos de valor numérico, este será el valor usado. En caso contrario, el valor será 0 (cero). Los datos numéricos válidos son un signo opcional, seguido por uno o más dígitos (que opcionalmente contengan un punto decimal), seguidos por un exponente opcional. El exponente es una 'e' o una 'E' seguidos por uno o más dígitos.

Cuando la primera expresión es una cadena, el tipo de la variable dependerá de la segunda expresión.

```
$foo = 1 + "10.5";                // $foo es doble (11.5)
$foo = 1 + "-1.3e3";              // $foo es doble (-1299)
$foo = 1 + "bob-1.3e3";           // $foo es entero (1)
$foo = 1 + "bob3";                // $foo es entero (1)
$foo = 1 + "10 Cerditos";         // $foo es entero (11)
$foo = "10.0 cerdos " + 1;        // $foo es entero (11)
$foo = "10.0 cerdos " + 1.0;      // $foo es double (11)
```

La siguiente línea le ayudará a comprobar los resultados en un script de PHP :

```
echo "\$foo==$foo; el tipo es " . gettype( $foo ) . "<br>\n";
```

Arreglos

En PHP los arreglos actualmente actúan tanto como tablas hash (arreglos asociativos) o como arreglos indexados (vectores).

Arreglos Unidimensionales

PHP soporta tanto arreglos escalares como asociativos. Se puede crear una arreglo usando las funciones **list()** o **array()**, o se puede asignar el valor de cada elemento del arreglo de manera explícita.

```
$a[0] = "abc";
$a[1] = "def";
$b[“foo”] = 13;
```

También se puede crear un arreglo simplemente añadiéndole valores a éste. Cuando se asigna un valor a una variable arreglo usando corchetes vacíos, el valor se añadirá al final del arreglo.

```
$a[] = "hola";   // $a[2] == "hola"
$a[] = "mundo";  // $a[3] == "mundo"
```

Los arreglos se pueden ordenar usando diversas funciones de PHP : **asort()**, **arsort()**, **ksort()**, **rsort()**, **sort()**, **uasort()** y **uksort()** dependiendo del tipo de ordenación requerida.

También se puede contar el número de elementos de un arreglo usando la función **count()** y recorrerlo usando las funciones **next()** y **prev()**. Otra forma habitual de recorrer un arreglo es usando la función **each()**.

Arreglos Multidimensionales

Los arreglos multidimensionales son bastante simples actualmente. Para cada dimensión del arreglo, se puede añadir otro valor [clave] al final :

```
$a[1]        = $f;         // ejemplos de una sola dimensión
$a["foo"]    = $f;
$a[1][0]     = $f;         // bidimensional
$a["foo"][2] = $f;         // (se pueden mezclar índices numéricos y asociativos)
$a[3]["bar"] = $f;         // (se pueden mezclar índices numéricos y asociativos)
$a["foo"][4]["bar"][0] = $f;   // tetradimensional!
```

Se pueden "llenar" arreglos multidimensionales de muchas formas, pero la más fácil es comprender cómo usar la función **array()** para arreglos asociativos. Estos dos ejemplos llenarán el arreglo unidimensional de la misma manera :

```
// Ejemplo 1:

$a["color"]  = "rojo";
$a["sabor"]  = "dulce";
$a["forma"]  = "redondeada";
$a["nombre"] = "manzana";
$a[3]        = 4;

// Ejemplo 2:

$a = array("color"=>"rojo","sabor"=>"dulce","forma"=>"redondeada","nombre"=>"manzana",3=>4);
```

Su representación gráfica podría ser la siguiente :

	$a
color	rojo
sabor	dulce
forma	redondeada
nombre	manzana
3	4

La función **array()** se puede usar para generar arreglos multidimensionales :

```
<?php

 $a = array("manzana" => array("color"=>"rojo","sabor"=>"dulce","forma"=>"redondeada"),
         "naranja" => array("color"=>"naranja","sabor"=>"ácido","forma"=>"redondeada"),
         "plátano" => array("color"=>"amarillo","sabor"=>"dulce","forma"=>"aplatanada"));

echo $a["manzana"]["sabor"];    // devolverá "dulce"

?>
```

Su representación gráfica podría ser la siguiente :

$a	color	sabor	forma
manzana	rojo	dulce	redondeada
naranja	naranja	ácido	redondeada
plátano	amarillo	dulce	aplatanada

Variables

En PHP las variables se representan como un signo de dólar seguido por el identificador de la variable. El nombre de la variable es sensible a minúsculas y mayúsculas.

```
$var = "Bob";
$Var = "Joe";
echo "$var, $Var"; // produce la salida "Bob, Joe"
```

PHP4 ofrece otra forma de asignar valores a las variables : *asignar por referencia*. Esto significa que la nueva variable simplemente referencia (en otras palabras, *"se convierte en un alias de"* o *"apunta a"*) a la variable original. Los cambios a la nueva variable afectan a la original, y viceversa. Esto también significa que no se produce una copia de valores; por tanto, la asignación ocurre más rápidamente. De cualquier forma, cualquier incremento de velocidad se notará sólo en los ciclos críticos cuando se asignen grandes arreglos u objetos.

Para asignar por referencia, simplemente se antepone un ampersand (&) al comienzo de la variable cuyo valor se está asignando (la variable fuente). Por ejemplo, el siguiente código produce la salida ' Mi nombre es Bob ' dos veces:

```
<?php

$foo = 'Bob';              // Asigna el valor 'Bob' a $foo
$bar = &$foo;              // Referencia $foo vía $bar.
$bar = "Mi nombre es $bar"; // Modificando a $bar...
echo $foo;                 // $foo también se modifica.
echo $bar;

?>
```

Algo importante a tener en cuenta es que sólo las variables con nombre pueden ser asignadas por referencia.

```
<?php

$foo = 25;
$bar = &$foo;      // Esta es una asignación válida.
$bar = &(24 * 7);  // Inválida; referencia a una expresión sin nombre.

function test() {
   return 25;
}

$bar = &test();    // Inválida.

?>
```

Ámbito de las variables

El ámbito de una variable es el contexto dentro del que la variable está definida. La mayor parte de las variables en PHP sólo tienen un ámbito simple. Este ámbito simple también abarca los archivos incluidos y los requeridos. Por ejemplo:

```
$a = 1;
include "b.inc";
```

Aquí, el ámbito de la variable $a abarca el script incluido en b.inc.

Dentro de las funciones definidas por el usuario aparece un ámbito local a la función. Cualquier variables que se use dentro de una función está, por defecto, limitada al ámbito local de esa función. Por ejemplo :

```
$a = 1; /* ámbito global */

Function Test () {
```

```
    echo $a; /* referencia a una variable de ámbito local */
}

Test ();
```

Este script no producirá salida, ya que la orden echo utiliza una versión local de la variable $a, a la que no se ha asignado ningún valor en su ámbito. Puede notarse una pequeña diferencia con el lenguaje C, en el que las variables globales están disponibles automáticamente dentro de la función a menos que sean expresamente sobre escritas por una definición local. Esto puede causar algunos problemas, ya que la gente puede cambiar variables globales inadvertidamente. En PHP, las variables globales deben ser declaradas globales dentro de la función si van a ser utilizadas dentro de ésta.

Ejemplo :

```
$a = 1;
$b = 2;

Function Sum () {
    global $a, $b;
    $b = $a + $b;
}

Sum ();
echo $b;
```

El script anterior producirá la salida "3". Al declarar $a y $b globales dentro de la función, todas las referencias a tales variables se referirán a la versión global. No hay límite al número de variables globales que se pueden manipular dentro de una función.

Un segundo método para acceder a las variables desde un ámbito global es usando el arreglo $GLOBALS propio de PHP3. El ejemplo anterior se puede rescribir así :

```
$a = 1;
$b = 2;

Function Sum () {
    $GLOBALS["b"] = $GLOBALS["a"] + $GLOBALS["b"];
}

Sum ();
echo $b;
```

El arreglo $GLOBALS es un arreglo asociativo con el nombre de la variable global como clave y los contenidos de dicha variable como el valor del elemento del arreglo.

Otra característica importante del ámbito de las variables es la variable static. Una variable estática existe sólo en el ámbito local de la función, pero no pierde su valor cuando la ejecución del programa abandona este ámbito. Consideremos el siguiente ejemplo:

```
Function Test () {
```

```
    $a = 0;
    echo $a;
    $a++;
}
```

Esta función tiene poca utilidad ya que cada vez que es llamada asigna a $a el valor. La sentencia $a++, que incrementa la variable, no sirve para nada, ya que en cuanto la función termina, la variable $a desaparece. Para hacer una función útil para contar, que no pierda la pista del valor actual del conteo, la variable $a debe declararse como estática:

```
Function Test () {
    static $a = 0;
    echo $a;
    $a++;
}
```

Ahora, cada vez que se llame a la función Test(), se preservará el valor de $a y se incrementará.

Las variables estáticas también proporcionan una forma de manejar funciones recursivas. Una función recursiva es la que se llama a sí misma. Se debe tener cuidado al escribir una función recursiva, ya que puede ocurrir que se llame a sí misma indefinidamente. Hay que asegurarse de implementar una forma adecuada de terminar la recursión. La siguiente función cuenta recursivamente hasta 10, usando la variable estática $count para saber cuándo parar:

```
Function Test () {
    static $count = 0;
    $count++;
    echo $count;

    if ($count < 10) {
        Test ();
    }

    $count--;
}
```

Variables Externas a PHP

Formularios HTML (GET y POST)

Cuando se envía un formulario a un script PHP, las variables de dicho formulario están disponibles automáticamente en el script gracias al mismo PHP. Por ejemplo, consideremos el siguiente formulario :

Variables de formulario simples

```
<form action="foo.php" method="post">
    Nombre: <input type="text" name="nombre"><br>
```

```
   <input type="submit">
</form>
```

Cuando el formulario sea enviado, PHP creará la variable $nombre, que contendrá lo que se introdujo en el campo Nombre : de dicho formulario.

PHP también maneja arreglos en el contexto de variables de formularios, pero sólo en una dimensión. Se puede, por ejemplo, agrupar juntas variables relacionadas, o usar esta característica para recuperar valores de un campo select input múltiple :

Variables de formulario más complejas

```
<form action="arreglo.php" method="post">
   Nombre : <input type="text" name="persona[name]"><br>
   Email : <input type="text" name="persona[email]"><br>
   Bebida : <br>
   <select multiple name="bebida[]">
      <option value="agua">Agua
      <option value="vino">Vino
      <option value="cerveza">Cerveza
   </select>
   <input type="submit">
</form>
```

Si la característica de PHP de track_vars está activada, ya sea mediante la opción de configuración track_vars o mediante la directiva <? php_track_vars ?>, las variables enviadas con los métodos POST o GET del formulario, también se encontrarán disponibles en los arreglos asociativos globales $HTTP_POST_VARS y $HTTP_GET_VARS.

Operadores

PHP brinda un conjunto de operadores muy sencillos de utilizar al formar expresiones.

Operadores Aritméticos

Ejemplo	Nombre	Resultado
$a + $b	Adición	Suma de $a y $b.
$a - $b	Substracción	Diferencia entre $a y $b.
$a * $b	Multiplicación	Producto de $a and $b.
$a / $b	División	Cociente de $a entre $b.
$a % $b	Módulo	Resto de $a dividido entre $b.

Operadores de Asignación

El operador básico de asignación es un signo igual " = ". A primera vista se puede pensar que es el operador de comparación "*igual que*". Pero no, realmente significa que el operando de la izquierda toma el valor de la expresión a la derecha, (esto es, " *toma el valor de* ").

El valor de una expresión de asignación es el propio valor asignado.

Esto es, el valor de $a = 3 es 3. Esto permite hacer cosas importantes como

$a = ($b = 4) + 5; // ahora $a es igual a 9, y $b vale 4.

Además del operador básico de asignación, existen los "operadores combinados" para todas las operaciones aritméticas y de cadenas que sean binarias. Este operador combinado permite, de una sola vez, usar una variable en una expresión y luego establecer el valor de esa variable al resultado de la expresión. Por ejemplo :

```
$a = 3;
$a += 5;       // Establece $a a 8, como si hubiésemos escrito: $a = $a + 5;
$b = "Hola ";
$b .= "Ahí!";  // Establece $b a "Hola Ahí!", igual que si hiciésemos $b = $b . "Ahí!";
```

Observe que la asignación realiza una nueva copia de la variable original (asignación por valor), por lo que cambios a la variable original no afectan a la copia. Esto puede tener interés si se requiere copiar algo como un arreglo con muchos elementos dentro de un ciclo que se repita muchas veces (cada vez se realizará una nueva copia del arreglo). PHP4 soporta asignación por referencia, usando la sintaxis $var = &$otravar;, pero esto no es posible en PHP3. Asignación por referencia quiere decir que ambas variables acabarán apuntando al mismo dato y que nada se copia realmente.

Operadores Bit a bit

Los operadores bit a bit permiten activar o desactivar bits individuales de un entero.

Ejemplo	**Nombre**	**Resultado**
$a & $b	Y	Se activan los bits que están activos tanto en $a como $b.
$a | $b	O	Se activan los bits que están activos en $a o que lo están en $b.
$a ^ $b	Xor ("o exclusiva")	Se activan los bits que están activos en $a o en $b pero no en ambos a la vez.
~ $a	No	Se activan los bits que no están activos en $a.
$a << $b	Desplazamiento a la izquierda	Desplaza los bits de $a, $b posiciones hacia la izquierda (por aritmética binaria, cada posición desplazada equivale a multiplicar por dos el valor de $a)
$a >> $b	Desplazamiento a la derecha	Desplaza los bits de $a, $b posiciones hacia la derecha (por aritmética binaria, cada posición desplazada equivale a dividir entre dos el valor de $a)

Operadores de Comparación

Los operadores de comparación, como su nombre lo indica, permiten comparar dos valores y establecer una relación entre éstos.

Ejemplo	Nombre	Resultado
$a == $b	Igualdad	Cierto si $a es igual a $b.
$a === $b	Identidad	Cierto si $a es igual a $b y si son del mismo tipo (sólo PHP4)
$a != $b	Desigualdad	Cierto si $a no es igual a $b.
$a < $b	Menor que	Cierto si $a es estrictamente menor que $b.
$a > $b	Mayor que	Cierto si $a es estrictamente mayor que $b.
$a <= $b	Menor o igual que	Cierto si $a es menor o igual que $b.
$a >= $b	Mayor o igual que	Cierto si $a es mayor o igual que $b.

Otro operador importante es el condicional o ternario representado por " ? : ", que funciona como en C y otros muchos lenguajes.

(expr1) ? (expr2) : (expr3);

La expresión toma el valor *expr2* si *expr1* se evalúa a **cierto**, y *expr3* si *expr1* se evalúa a **falso**.

Operadores de Incremento / Decremento

PHP soporta los operadores de pre decremento y post incremento al estilo de C.

Ejemplo	Nombre	Efecto
++$a	Preincremento	Incrementa $a en uno y después devuelve $a.
$a++	Postincremento	Devuelve $a y después incrementa $a en uno.
--$a	Predecremento	Decrementa $a en uno y después devuelve $a.
$a--	Postdecremento	Devuelve $a y después decrementa $a en uno.

Ejemplo :

```
<?php
echo "<h3>Postincremento</h3>";
$a = 5;
echo "Debe ser 5: ".$a++."<br>\n";
echo "Debe ser 6: ".$a."<br>\n";
echo "<h3>Preincremento</h3>";
```

```
$a = 5;
echo "Debe ser 6: ".++$a."<br>\n";
echo "Debe ser 6: ".$a."<br>\n";
echo "<h3>Postdecremento</h3>";

$a = 5;
echo "Debe ser 5: ".$a--."<br>\n";
echo "Debe ser 4: ".$a. "<br>\n";
echo "<h3>Predecremento</h3>";

$a = 5;
echo "Debe ser 4: ".--$a. "<br>\n";
echo "Debe ser 4: ".$a. "<br>\n";

?>
```

Operadores Lógicos

Ejemplo	Nombre	Resultado
$a and $b	Y	Cierto si tanto $a como $b son ciertos.
$a or $b	O	Cierto si $a o $b son ciertos.
$a xor $b	O exclusiva	Cierto si $a es cierto o $b es cierto, pero no ambos a la vez.
! $a	Negación	Cierto si $a no es cierto.
$a && $b	Y	Cierto si tanto $a como $b son ciertos.
$a || $b	O	Cierto si $a o $b son ciertos.

La razón de las dos variaciones de "y" y "o" es que operan con distinta precedencia

Precedencia de Operadores

La precedencia de operadores especifica cómo se agrupan las expresiones. Por ejemplo, en la expresión 1 + 5 * 3, la respuesta es 16 y no 18 porque el operador de multiplicación (" * ") tiene una mayor precedencia que el de adición (" + ").

La siguiente tabla lista la precedencia de operadores, indicándose primero los de menor precedencia.

izquierda	,
izquierda	or
izquierda	xor
izquierda	and

izquierda	,
derecha	print
izquierda	= += -= *= /= .= %= &= |= ^= ~= <<= >>=
izquierda	? :
izquierda	||
izquierda	&&
izquierda	|
izquierda	^
izquierda	&
no asociativo	== != ===
no asociativo	< <= > >=
izquierda	<< >>
izquierda	+ - .
izquierda	* / %
derecha	! ~ ++ -- (int) (double) (string) (array) (object) @
derecha	[
no asociativo	new

Operadores de Cadenas

Hay dos operadores de cadenas. El primero es el operador de concatenación punto (.), que devuelve el resultado de concatenar sus operandos izquierdo y derecho. El segundo es el operador de concatenación y asignación punto igual (.=).

```
$a = "Hola ";
$b = $a . "Mundo!"; // ahora $b contiene "Hola Mundo!"
$a = "Hola ";
$a .= "Mundo!";     // ahora $a contiene "Hola Mundo!"
```

Sentencias

Todo archivo de comandos PHP se compone de una serie de sentencias. Una sentencia puede ser una asignación, una llamada a función, un ciclo, una sentencia condicional e incluso una sentencia que no haga nada (una sentencia vacía). Las sentencias normalmente acaban con punto y coma. Además, las sentencias pueden contener grupos de sentencias encerradas entre llaves. Un grupo de sentencias es también una sentencia.

Estructura de decisión if

La construcción if es una de las más importantes características de muchos lenguajes, incluido PHP. Permite la ejecución condicional de fragmentos de código. PHP define una estructura if que es similar a la de C:

```
if (expr)
  sentencia
```

Donde expr se evalúa y utiliza como valor condicional. Si expr evalúa a verdadero (**TRUE)**, PHP ejecutará la sentencia y si se evalúa a falso (**FALSE**) la ignorará.

El siguiente ejemplo mostraría ***a es mayor que b*** si $a fuera mayor que $b :

```
if ($a > $b)
  print "a es mayor que b";
```

A menudo, se desea tener más de una sentencia ejecutada de forma condicional. Por supuesto, no hay necesidad de encerrar cada sentencia con una cláusula if. En vez de eso, se pueden agrupar varias sentencias. Por ejemplo, este código mostraría ***a es mayor que b*** si $a fuera mayor que $b, y entonces asignaría el valor de $a a $b :

```
if ($a > $b) {
  print "a es mayor que b";
  $b = $a;
}
```

Las sentencias if se pueden anidar indefinidamente dentro de otras sentencias if, lo cual proporciona una flexibilidad completa para ejecuciones condicionales en las diferentes partes del programa.

else

A menudo queremos ejecutar una sentencia si se cumple una cierta condición y otra sentencia distinta si la condición no se cumple. Esto es para lo que sirve else. El complemento else extiende una sentencia if para ejecutar una sentencia en caso de que la expresión en la sentencia if se evalúe como **falsa**. Por ejemplo, el siguiente código mostraría ***a es mayor que b*** si $a fuera mayor que $b, y ***a NO es mayor que b*** en cualquier otro caso :

```
if ($a > $b) {
  print "a es mayor que b";
}
else {
```

```
   print "a NO es mayor que b";
}
```

elseif

elseif, como su nombre lo sugiere, es una combinación de if y else. Como else, extiende una sentencia if para ejecutar una sentencia diferente en caso de que la expresión if original sea evaluada como **falsa**. No obstante, a diferencia de else, se ejecutará expresión alternativa solamente si la expresión condicional elseif se evalúa como **TRUE**. Por ejemplo, el siguiente código mostraría ***a es mayor que b***, ***a es igual a b*** o ***a es menor que b***:

```
if ($a > $b) {
   print "a es mayor que b";
} elseif ($a == $b) {
   print "a es igual que b";
} else {
   print "a es mayor que b";
}
```

Puede haber varios elseif's dentro de la misma sentencia if. La primera expresión elseif (si hay alguna) que se evalúe como **verdadero** se ejecutaría. En PHP, también se puede escribir else if (con dos palabras) y el comportamiento sería idéntico al de un elseif (una sola palabra). El significado sintáctico es ligeramente distinto pero la línea básica es que ambos resultarían tener exactamente el mismo comportamiento.

La sentencia elseif se ejecuta sólo si la expresión if precedente y cualquier expresión elseif precedente se evalúan como **falsas** y la expresión elseif actual se evalúa como **verdadera**.

Estructura cíclica while

Los ciclos while son los más simples en PHP. Se comportan como su contrapartida en C. La forma básica de una sentencia while es :

while (expr) sentencia

El significado de una sentencia while es simple. Le dice a PHP que ejecute la(s) sentencia(s) anidada(s) repetidamente, mientras la expresión while se evalúe como **verdadera**. El valor de la expresión es comprobado cada vez al principio del ciclo, así que incluso si este valor cambia durante la ejecución de la(s) sentencia(s) anidada(s), la ejecución no parará hasta el fin de la iteración (*cada vez que PHP ejecuta las sentencias en el ciclo es una iteración*). A veces, si la expresión while se evalúa como **falso** desde el principio de todo, la(s) sentencia(s) anidada(s) no se ejecutarán ni siquiera una vez.

Como con la sentencia while, se pueden agrupar múltiples sentencias dentro del mismo ciclo while, encerrando un grupo de sentencias con llaves, o usando la sintaxis alternativa :

while (expr): sentencia ... endwhile;

Los siguientes ejemplos son idénticos y ambos imprimen números del 1 al 10:

```
/* ejemplo 1 */
$i = 1;

while ($i <= 10) {
    print $i++;  /* el valor impreso sería $i antes del incremento ( post-incremento ) */
}

/* ejemplo 2 */

$i = 1;

while ($i <= 10):
    print $i;
    $i++;
endwhile;
```

Estructura cíclica do..while

Los ciclos do..while son muy similares a los ciclos while, excepto que las condiciones se comprueban al final de cada iteración en vez de al principio. La principal diferencia frente a los ciclos regulares while es que se garantiza la ejecución de la primera iteración de un ciclo do..while (la condición se comprueba sólo al final de la iteración), mientras que puede no ser necesariamente ejecutada con un ciclo while regular (la condición se comprueba al principio de cada iteración, si esta se evalúa como **falsa** desde el principio la ejecución del ciclo finalizará inmediatamente).

Hay una sola sintaxis para los ciclos do..while :

```
$i = 0;
do {
   print $i;
}while ($i>0);
```

El ciclo de arriba se ejecutaría exactamente una sola vez, después de la primera iteración, cuando la condición se comprueba, se evalúa como **falsa** ($i no es más grande que 0) y la ejecución del ciclo finaliza.

Los usuarios avanzados de C pueden estar familiarizados con un uso distinto del ciclo do..while, que permite detener la ejecución en medio del bloques de código, encapsulándolo con un do..while(0) y usando la sentencia break para salir. El siguiente fragmento de código demuestra esto :

```
do {
    if ($i < 5) {
        print "i no es lo suficientemente grande";
        break;
    }
    $i *= $factor;
    if ($i < $minimum_limit) {
        break;
    }
```

```
    print "i es correcto";
    ...procesa i...
} while(0);  // Ciclo infinito controlado con break
```

Estructura cíclica for

Los ciclos for son más complejos en PHP. Se comportan como su contrapartida en C. La sintaxis de un ciclo for es :

for (expr1; expr2; expr3) sentencia

La primera expresión (*expr1*) se evalúa o ejecuta incondicionalmente una vez al principio del ciclo.

Al comienzo de cada iteración, se evalúa *expr2* . Si se evalúa como **verdadera**, el ciclo continúa y las sentencias anidadas se ejecutan. Si se evalúa como **falsa**, la ejecución del ciclo finaliza. Al final de cada iteración, se evalúa *expr3*.

Cada una de las expresiones puede estar vacía. Que *expr2* esté vacía significa que el ciclo debería correr indefinidamente (PHP implícitamente lo considera como **verdadero**, al igual que C). Esto puede que no sea tan inútil como se podría pensar, puesto que a menudo se quiere salir de un ciclo usando una sentencia break condicional en vez de usar la condición del for.

Considere los siguientes ejemplos. Todos ellos muestran números del 1 al 10:

```
/* ejemplo 1 */
for ($i = 1; $i <= 10; $i++) {
    print $i;
}

/* ejemplo 2 */

for ($i = 1;;$i++) {
    if ($i > 10) {
        break;
    }
    print $i;
}

/* ejemplo 3 */
$i = 1;

for (;;) {
    if ($i > 10) {
        break;
    }
    print $i;
    $i++;
}
```

```
/* ejemplo 4 */

for ($i = 1; $i <= 10; print $i, $i++) ;
```

PHP también soporta la "***sintaxis de dos puntos***" alternativa para ciclos for.

```
for (expr1; expr2; expr3): sentencia; ...; endfor;
```

break

break cambia incondicionalmente el flujo de ejecución hacia afuera de las siguientes estructuras : for, while, o switch.

break accepta un parámetro opcional, el cual determina de cuántas estructuras de control debe que salir.

continue

continue se usa dentro de la estructura del ciclo para saltar el resto de la iteración actual en éste y continuar la ejecución al comienzo de la siguiente iteración.

continue accepta un parámetro opcional, el cual determina cuántos niveles de ciclos debe que saltar antes de continuar con la ejecución.

Estructura de decisión switch

La sentencia switch es similar a una serie de sentencias if en la misma expresión. En muchas ocasiones, se quiere comparar la misma variable (o expresión) con muchos valores diferentes y ejecutar una parte de código distinta dependiendo de a qué valor sea igual. Para ello sirve la sentencia switch.

Los siguientes dos ejemplos son dos modos distintos de escribir la misma estructura, uno usa una serie de sentencias if y el otro usa la sentencia switch :

```
if ($i == 0) {
   print "i es igual a 0";
}

if ($i == 1) {
   print "i es igual a 1";
}

if ($i == 2) {
   print "i es igual a 2";
}

switch ($i) {
   case 0:
       print "i es igual a 0";
```

```
        break;
    case 1:
        print "i es igual a 1";
        break;
    case 2:
        print "i es igual a 2";
       break;
}
```

Es importante entender cómo se ejecuta la sentencia switch para evitar errores. La sentencia switch se ejecuta línea por línea (realmente, sentencia a sentencia). Al comienzo, no se ejecuta código. Sólo cuando se encuentra una sentencia case con un valor que coincide con el valor de la expresión switch. y PHP comienza a ejecutar las sentencias. PHP seguirá ejecutando las sentencias hasta el final del bloque switch o hasta la primer sentencia break que encuentre. Si hay una sentencia break al final de una lista de sentencias case, PHP seguirá ejecutando las sentencias del siguiente case. Por ejemplo :

```
switch ($i) {
    case 0:
        print "i es igual a 0";
    case 1:
        print "i es igual a 1";
    case 2:
        print "i es igual a 2";
}
```

Aquí, si $i es igual a 0, *¡PHP ejecutaría todas las sentencias* print*!.* Si $i es igual a 1, PHP ejecutaría las últimas dos sentencias print y sólo si $i es igual a 2, se obtendría la conducta ***'esperada'*** y solamente se mostraría '***i es igual a 2'***. Así, es importante no olvidar las sentencias break (incluso aunque pueda querer evitar escribirlas intencionalmente en ciertas circunstancias).

En una sentencia switch, la condición se evalúa sólo una vez y el resultado se compara en cada sentencia case. En una sentencia elseif, la condición se evalúa otra vez. Si la condición es más compleja que una comparación simple y se está dentro de un ciclo corto, una estructura switch puede ser más conveniente.

La lista de sentencias de un case puede también estar vacía, lo cual simplemente pasa el control a la lista de sentencias del siguiente case.

```
switch ($i) {
    case 0:
    case 1:
    case 2:
        print "i es menor que 3, pero no negativo";
        break;
    case 3:
        print "i es 3";
}
```

Un caso especial es el default. Este tipo de case coincide con todo lo que no coincidieron los otros case. Por ejemplo:

```
switch ($i) {
    case 0:
        print "i es igual a 0";
        break;
    case 1:
        print "i es igual a 1";
        break;
    case 2:
        print "i es igual a 2";
        break;
    default:
        print "i no es igual a 0, 1 o 2";
}
```

La expresión case puede ser cualquier expresión que se evalúe a un tipo simple, es decir, números enteros, de punto flotante o cadenas de texto. No se puede usar aquí ni arreglos ni objetos a menos que se conviertan a un tipo simple.

La sintaxis alternativa para las estructuras de control, antes mencionada, está disponible para el switch.

```
switch ($i):
    case 0:
        print "i es igual 0";
        break;
    case 1:
        print "i es igual a 1";
        break;
    case 2:
        print "i es igual a 2";
        break;
   default:
   print "i no es igual a 0, 1 o 2";
endswitch;
```

Funciones definidas por el usuario

Una función se define con la siguiente sintaxis :

```
function foo ($arg_1, $arg_2, ..., $arg_n) {
  echo "Función de ejemplo.\n";
  return $retval;
}
```

Pasar parámetros por referencia

Por defecto, los parámetros de una función se pasan por valor (de manera que si cambias el valor del argumento dentro de la función, no se ve modificado fuera de ella). Si desea permitir a una función modificar sus parámetros, se deben pasar por referencia. Esto se logra anteponiendo un *ampersand* (&) al nombre del parámetro en la definición de la función:

```
function add_some_extra(&$string) {
    $string .= ' y algo más.';
}

$str = 'Esto es una cadena, ';
add_some_extra($str);
echo $str;    // Saca 'Esto es una cadena, y algo más.'
```

Funciones MySQL

PHP brinda un conjunto muy amplio de funciones que permiten acceder a servidores de bases de datos utilizando MySQL.

Puede encontrar más información sobre MySQL en http://www.mysql.com/

A continuación se presentan las funciones más utilizadas en PHP para manejar una base de datos con MySQL. Ejemplos completos de su uso se analizarán en la siguiente sección del curso.

Tabla de Funciones MySQL

Función	Descripción
mysql_affected_rows	Devuelve el número de filas afectadas de la última operación MySQL
mysql_change_user	Cambia el usuario conectado en la conexión activa
mysql_client_encoding	Devuelve el nombre del juego de caracteres
mysql_close	Cierra el enlace con MySQL
mysql_connect	Abre una conexión a un servidor MySQL
mysql_create_db	Crea una base MySQL
mysql_data_seek	Mueve el puntero interno
mysql_db_name	Obtener datos de resultado
mysql_db_query	Envía una sentencia MySQL al servidor
mysql_drop_db	Borra una base de datos MySQL

mysql_error	Devuelve el texto del mensaje de error de la última operación MySQL
mysql_escape_string	Escapa una cadena para su uso en mysql_query.
mysql_fetch_array	Extrae la fila de resultado como una matriz asociativa
mysql_fetch_assoc	Recupera una fila de resultado como una matriz asociativa
mysql_fetch_field	Extrae la información de una columna y la devuelve como un objeto.
mysql_fetch_lengths	Devuelve la longitud de cada salida en un resultado
mysql_fetch_object	Extrae una fila de resultado como un objeto
mysql_fetch_row	Devuelve una fila de resultado como matriz
mysql_field_flags	Devuelve los flags asociados con el campo especificado en un resultado
mysql_field_len	Devuelve la longitud del campo especificado
mysql_field_name	Devuelve el nombre del campo especificado en un resultado
mysql_field_seek	Asigna el puntero del resultado al offset del campo especificado
mysql_field_table	Devuelve el nombre de la tabla donde esta el campo especificado
mysql_field_type	Devuelve el tipo del campo especificado en un resultado
mysql_free_result	Libera la memoria del resultado
mysql_get_client_info	Obtener información del cliente MySQL
mysql_get_host_info	Obtener información de la máquina anfitriona MySQL
mysql_get_proto_info	Obtener información del protocolo MySQL
mysql_get_server_info	Obtener información del servidor MySQL
mysql_info	Obtiene información sobre la consulta más reciente
mysql_insert_id	Devuelve el identificador generado en la última llamada a INSERT
mysql_list_dbs	Lista las bases de datos disponibles en el servidor MySQL
mysql_list_fields	Lista los campos del resultado de MySQL
mysql_list_processes	Lista los procesos MySQL
mysql_list_tables	Lista las tablas en una base de datos MySQL
mysql_num_fields	Devuelve el numero de campos de un resultado

mysql_num_rows	Devuelve el numero de filas de un resultado
mysql_pconnect	Abre una conexión persistente al servidor MySQL
mysql_ping	Efectuar un chequeo de respuesta (ping) sobre una conexión de servidor o reconectarse si no hay conexión
mysql_query	Envía una sentencia SQL a MySQL
mysql_real_escape_string	Escapa caracteres especiales de una cadena para su uso en una sentencia SQL, tomando en cuanto el juego de caracteres actual de la conexión.

¿Qué es JavaScript realmente?

JavaScript es un robusto lenguaje de programación que se puede aplicar a un documento HTML y usarse para crear interactividad dinámica en los sitios web. Fue inventado por Brendan Eich, cofundador del proyecto Mozilla, Mozilla Foundation y la Corporación Mozilla.
Puedes hacer casi cualquier cosa con JavaScript. Puedes empezar con pequeñas cosas como carruseles, galerías de imágenes, diseños fluctuantes, y respuestas a las pulsaciones de botones. Con más experiencia, serás capaz de crear juegos, animaciones 2D y gráficos 3D, aplicaciones integradas basadas en bases de datos ¡y mucho más!

JavaScript por sí solo es bastante compacto aunque muy flexible, y los desarrolladores han escrito gran cantidad de herramientas encima del núcleo del lenguaje JavaScript, desbloqueando una gran cantidad de funcionalidad adicional con un mínimo esfuerzo. Esto incluye:

- Interfaces de Programación de Aplicaciones del Navegador (APIs) — APIs construidas dentro de los navegadores que ofrecen funcionalidades como crear dinámica mente contenido HTML y establecer estilos CSS, hasta capturar y manipular un vídeo desde la cámara web del usuario, o generar gráficos 3D y muestras de sonido.
- APIs de terceros, que permiten a los desarrolladores incorporar funcionalidades en sus sitios de otros proveedores de contenidos como Twitter o Facebook.
- Marcos de trabajo y librerías de terceros que puedes aplicar a tu HTML para que puedas construir y publicar rápidamente sitios y aplicaciones.

Ya que se supone que este artículo es solo una introducción ligera a JavaScript, la intención no es confundirte en esta etapa hablando en detalle sobre cuál es la diferencia entre el núcleo del lenguaje JavaScript y las diferentes herramientas listadas arriba. Puedes aprender todo eso en detalle más tarde, en el Área de Aprendizaje en MDN, y en el resto de MDN.
Debajo se presentan algunos aspectos del núcleo del lenguaje y también jugarás con unas pocas características de la API del navegador. ¡Diviértete!

Ejemplo «¡Hola Mundo!»

La sección de arriba suena realmente emocionante, y debería serlo. JavaScript es una de las tecnologías web más emocionantes, y cuando comiences a ser bueno en su uso, tus sitios web entrarán en una nueva dimensión de energía y creatividad.

Sin embargo, sentirse cómodo con JavaScript es un poco más difícil que sentirse cómodo con HTML y CSS. Deberás comenzar poco a poco y continuar trabajando en pasos pequeños y consistentes. Para comenzar, mostraremos cómo añadir JavaScript básico a tu página, creando un «¡Hola Mundo!» de ejemplo (el estándar en los ejemplos básicos de programación).

```
const miTitulo = document.querySelector('h1');
miTitulo.textContent = '¡Hola mundo!';
```

Finalmente, asegúrate de que has guardado los archivos HTML y JavaScript, y abre index.html en el navegador. Deberías ver algo así:

¿Qué ha ocurrido?

El texto del título ha sido cambiado por ¡Hola mundo! usando JavaScript. Hiciste esto primero usando la función querySelector() para obtener una referencia al título y almacenarla en una variable llamada miTitulo. Esto es muy similar a lo que hiciste con CSS usando selectores —quieres hacer algo con un elemento, así que tienes que seleccionarlo primero—.
Después de eso, estableciste el valor de la propiedad textContent de la variable miTitulo (que representa el contenido del título) como ¡Hola mundo!

Curso intensivo de fundamentos del lenguaje

Ahora se explicarán algunas de las funciones básicas del lenguaje JavaScript para que puedas comprender mejor cómo funciona todo. Mejor aún, estas características son comunes para todos los lenguajes de programación. Si puedes entender esos fundamentos, deberías ser capaz de comenzar a programar en casi cualquier cosa.

Variables

Las Variables son contenedores en los que puedes almacenar valores. Primero debes declarar la variable con la palabra clave var (menos recomendado) o let, seguida del nombre que le quieras dar. Se recomienda más el uso de let que de var (más adelante se profundiza un poco sobre esto):

let nombreDeLaVariable;

Nota: todas las líneas en JS deben acabar en punto y coma (;) para indicar que es ahí donde termina la declaración. Si no los incluyes puedes obtener resultados inesperados. Sin embargo, algunas personas creen que es una buena práctica tener punto y coma al final de cada declaración. Hay otras reglas para cuando se debe y no se debe usar punto y coma. Para más detalles, vea Guía del punto y coma en JavaScript (en inglés).
Nota: puedes llamar a una variable con casi cualquier nombre, pero hay algunas restricciones (ver este artículo sobre las reglas existentes). Si no estás seguro, puedes comprobar el nombre de la variable para ver si es válido.
Nota: JavaScript distingue entre mayúsculas y minúsculas. miVariable es una variable distinta a mivariable. Si estás teniendo problemas en tu código, revisa las mayúsculas y minúsculas.
Nota: para más detalles sobre la diferencia entre var y let, vea Diferencia entre var y let.

Tras declarar una variable, puedes asignarle un valor:

```
nombreDeLaVariable = 'Bob';
```

Puedes hacer las dos cosas en la misma línea si lo necesitas:

```
let nombreDeLaVariable = 'Bob';
```

Puedes obtener el valor de la variable llamándola por su nombre:

```
nombreDeLaVariable;
```

Después de haberle dado un valor a la variable, puedes volver a cambiarlo:

```
let nombreDeLaVariable = 'Bob';
nombreDeLaVariable = 'Steve';
```

Advierte que las variables tienen distintos tipos de datos:

Variable	Explicación	Ejemplo
String	Esto es una secuencia de texo conocida como cadena. Para indicar que la variable es una cadena, debes escribirlo entre comillas.	let miVariable = 'Bob';
Number	Esto es un número. Los números no tienen comillas.	let miVariable = 10;
Boolean	Tienen valor verdadero/falso. true/false son palabras especiales en JS, y no necesitan comillas.	let miVariable = true;
Array	Una estructura que te permite almacenar varios valores en una sola referencia.	let miVariable = [1,'Bob','Steve',10]; Llama a cada miembro del array así: miVariable[0], miVariable[1], etc.
Object	Básicamente cualquier cosa. Todo en JavaScript es un objeto y puede ser almacenado en una variable. Mantén esto en mente mientras aprendes.	let miVariable = document.querySelector('h1'); Todos los ejemplos anteriores también.

Entonces, ¿para qué necesitamos las variables? Las variables son necesarias para hacer cualquier cosa interesante en programación. Si los valores no pudieran cambiar, entonces no podrías hacer nada dinámico, como personalizar un mensaje de bienvenida de un usuario que visita tu página, cambiar la imagen que se muestra en una galería de imágenes, etc.

Comentarios

Puedes escribir comentarios entre el código JavaScript, igual que puedes en CSS. El navegador ignora el texto marcado como comentario. En JavaScript, los comentarios de una sola línea se escriben así:

```
// Esto es un comentario
```

Pero también puedes escribir comentarios en más de una línea, igual que en CSS:

```
/*
Esto es un comentario
de varias líneas.
*/
```

Operadores

Un operador es básicamente un símbolo matemático que puede actuar sobre dos valores (o variables) y producir un resultado. En la tabla de abajo aparecen los operadores más simples, con algunos ejemplos para probarlos en la consola del navegador.

Operador	Explicación	Símbolo(s)	Ejemplo
Suma/concatena	Se usa para sumar dos números, o juntar dos cadenas en una.	+	6 + 9; "Hola " + "mundo!";

Resta, multiplicación, división	Estos hacen lo que esperarías que hicieran en las matemáticas básicas.	-, *, /	9 - 3;
			8 * 2; // La multiplicación en JS es un asterisco
			9 / 3;

Operador de asignación	Los has visto anteriormente: asigna un valor a una variable.	=	let miVariable = 'Bob';
identidad/ igualdad	Comprueba si dos valores son iguales entre sí, y devuelve un valor de true/false (booleano).	===	let miVariable = 3; miVariable === 4;

Negación, distinto (no igual)	En ocasiones utilizado con el operador de identidad, la negación es en JS el equivalente al operador lógico NOT — cambia true por false y viceversa.	!, !==	La expresión básica es true, pero la comparación devuelve false porque lo hemos negado:
			let miVariable = 3;
			!miVariable === 3;
			Aquí estamos comprobando "miVariable NO es igual a 3". Esto devuelve false, porque miVariable ES igual a 3.
			let miVariable = 3;
			miVariable !== 3;

Hay muchos operadores por explorar, pero con esto será suficiente por ahora. Mira Expresiones y operadores para ver la lista completa.

Nota: mezclar tipos de datos puede dar lugar a resultados extraños cuando se hacen cálculos, así que asegúrate de que relacionas tus variables correctamente y de que recibes los resultados que esperabas. Por ejemplo, teclea: "3" + "25" en tu consola. ¿Por qué no obtienes lo que esperabas? Porque las comillas convierten los números en "strings" (el término inglés para denominar cadenas de caracteres) y de este modo has acabado con los "strings" concatenados entre sí, y no con los números sumados. Si tecleas: 35 + 25, obtendrás el resultado correcto.

Condicionales

Las condicionales son estructuras de código que permiten comprobar si una expresión devuelve true o no, y después ejecuta un código diferente dependiendo del resultado. La forma de condicional más común es la llamada if... else. Entonces, por ejemplo:

```
let helado = 'chocolate';
if (helado === 'chocolate') {
 alert('¡Sí, amo el helado de chocolate!');
} else {
 alert('Awwww, pero mi favorito es el de chocolate...');
}
```

La expresión dentro de if (...) es el criterio — este usa al operador de identidad (descrito arriba) para comparar la variable helado con la cadena chocolate para ver si las dos son iguales. Si esta comparación devuelve true, el primer bloque de código se ejecuta. Si no, ese código se omite y se ejecuta el segundo bloque de código después de la declaración else.

Funciones

Las funciones son una manera de encapsular una funcionalidad que quieres reutilizar, de manera que puedes llamar esa función con un solo nombre, y no tendrás que escribir el código entero cada vez que la utilices. Ya has visto algunas funciones más arriba, por ejemplo:

```
1.let nombreDeLaVariable = document.querySelector('h1');
2.alert('¡Hola!');
```

Estas funciones document.querySelector y alert están integradas en el navegador para poder utilizarlas en cualquier momento.
Si ves algo que parece un nombre de variable, pero tiene paréntesis —()— al final, probablemente es una función. Las funciones con frecuencia toman argumentos —pedazos de datos que necesitan para hacer su trabajo—. Estos se colocan dentro de los paréntesis, y se separan con comas si hay más de uno.
Por ejemplo, la función alert() hace aparecer una ventana emergente dentro de la ventana del navegador, pero necesitas asignarle una cadena como argumento para decirle qué mensaje se debe escribir en la ventana emergente.
Las buenas noticias son que podemos definir nuestras propias funciones —en el siguiente ejemplo escribimos una función simple que toma dos números como argumentos y los multiplica entre sí—:

```
function multiplica(num1,num2) {

 let resultado = num1 * num2;
 return resultado;

}
```

Trata de ejecutar la función anterior en la consola. Después trata de usar la nueva función algunas veces, p.ej:

```
multiplica(4, 7);
multiplica(20, 20);
multiplica(0.5, 3);
```

Nota: la sentencia return le dice al navegador que devuelva la variable resultado fuera de la función, para que esté disponible para su uso. Esto es necesario porque las variables definidas dentro de funciones, solo están disponibles dentro de esas funciones. Esto se conoce como «ámbito (scope en inglés) de la variable». Lee más sobre ámbito o alcance de la variable.

Eventos

Para crear una interacción real en tu sitio web, debes usar eventos. Estos son unas estructuras de código que captan lo que sucede en el navegador, y permite que en respuesta a las acciones que suceden se ejecute un código. El ejemplo más obvio es un clic (click event), que se activa al hacer clic sobre algo. Para demostrar esto, prueba ingresando lo siguiente en tu consola, luego da clic sobre la página actual:

```
document.querySelector('html').onclick = function() {
   alert('¡Ouch! ¡Deja de pincharme!');
}
```

Hay muchas maneras de enlazar un evento a un elemento; aquí hemos seleccionado el elemento <html> y le asignamos a su propiedad onclick una función anónima (función sin nombre) que contiene el código que se ejecutará cuando el evento suceda.
Nota que

```
document.querySelector('html').onclick = function(){};
```

es equivalente a

```
let miHTML = document.querySelector('html');
miHTML.onclick = function(){};
```

es solo un modo más corto de escribirlo.

Sobrecargar tu sitio web de ejemplo

Ahora vas a repasar un poco lo básico de JavaScript. Añadirás un par de funcionalidades a tu sitio para demostrar lo que puedes hacer.

Añadir un cambiador de imagen

En esta sección añadirás otra imagen a tu sitio usando la DOM API y agregarás un poco de código para cambiar entre imágenes al hacer clic.

1. Primero que todo, busca una imagen que te guste para tu sitio. Asegúrate que sea del mismo tamaño que la primera, o lo más cerca posible.
2. Guarda tu imagen en tu carpeta images.
3. Renombra esta imagen «firefox2.png» (sin las comillas).
4. Ve a tu archivo main.js y agrega el siguiente JavaScript (si tu JavaScript de «Hola Mundo» está aún allí, bórralo).

```
let miImage = document.querySelector('img');
miImage.onclick = function () {

  let miSrc = miImage.getAttribute('src');
  if (miSrc === 'images/firefox-icon.png') {

    miImage.setAttribute('src','images/firefox2.png');

  } else {

    miImage.setAttribute('src', 'images/firefox-icon.png');

  }

}
```

5.Guarda todos los archivos y carga index.html en tu navegador. Ahora cuando hagas clic en la imagen, ¡esta debe cambiar por otra!
Esto fue lo que sucedió: se almacena una referencia a tu elemento <img> en la variable miImage. Luego, haces que esta propiedad del manejador de evento onclick de la variable sea igual a una función sin nombre (una función «anónima»). Ahora, cada vez que se haga clic en la imagen:
1.El código recupera el valor del atributo src de la imagen.
2.El código usa una condicional para comprobar si el valor src es igual a la ruta de la imagen original:
1.Si es así, el código cambia el valor de src a la ruta de la segunda imagen, forzando a que se cargue la otra imagen en el elemento <img>.
2.Si no es así (significa que ya fue modificada), se cambiará el valor de src nuevamente a la ruta de la imagen original, regresando a como era en un principio.

Añadir un mensaje de bienvenida personalizado

Ahora añadirás un poco más de código, para cambiar el título de la página o incluir un mensaje personalizado de bienvenida para cuando el usuario ingrese por primera vez. Este mensaje de bienvenida permanecerá luego de que el usuario abandone la página y estará disponible para cuando regrese. Lo guardarás usando Web Storage API. También se incluirá una opción para cambiar el usuario y por lo tanto también el mensaje de bienvenida en cualquier momento que se requiera.
1.En index.html, agrega el siguiente código antes del elemento <script>:

```
<button>Cambiar de usuario</button>
```

2.En main.js, agrega el siguiente código al final del archivo, exactamente como está escrito. Esto toma referencia al nuevo botón que se agregó y al título y los almacena en variables:

```
let miBoton = document.querySelector('button');
let miTitulo = document.querySelector( 'h1');
```

3.Ahora agrega la siguiente función para poner el saludo personalizado, lo que no causará nada aún, pero arreglarás esto en un momento:

```
function estableceNombreUsuario() {

  let miNombre = prompt('Por favor, ingresa tu nombre.');
  localStorage.setItem('nombre', miNombre);

  miTitulo.textContent = 'Mozilla es genial,' + miNombre;

}
```

La función estableceNombreUsuario() contiene una función prompt(), que crea un cuadro de diálogo como lo hace alert(); la diferencia es que prompt() pide al usuario un dato, y almacena este dato en una variable cuando el botón Aceptar del cuadro de diálogo es presionado. En este caso, pedirás al usuario que ingrese su nombre. Luego, llamarás la API localStorage, que nos permite almacenar datos en el navegador y recuperarlos luego. Usarás la función setItem() de localStorage, que crea y

almacena un dato en el elemento llamado 'nombre', y coloca este valor en la variable miNombre que contiene el nombre que el usuario ingresó. Finalmente, establecerás el textContent del título a una cadena, más el nombre de usuario recientemente almacenado.

4. Luego, agregarás este bloque if ... else. Se podría llamar a esto el código de inicialización, como se ha establecido para cuando carga la app por primera vez:

```
if (!localStorage.getItem('nombre')) {

  estableceNombreUsuario();

}

else {

  let nombreAlmacenado = localStorage.getItem('nombre');
  miTitulo.textContent = 'Mozilla es genial,' + nombreAlmacenado;

}
```

La primera línea de este bloque usa el operador de negación (NO lógico representado por !) para comprobar si el elemento 'nombre' existe. Si no existe, la función estableceNombreUsuario() se iniciará para crearlo. Si ya existe (como por ejemplo cuando el usuario ya ingresó al sitio), se recupera el dato del nombre usando getItem() y se fija mediante textContent del título a la cadena, más el nombre del usuario, como hiciste dentro de estableceNombreUsuario().

5. Finalmente, agrega abajo el evento onclick que manipulará el botón, de modo que cuando sea pulsado se inicie la función estableceNombreUsuario(). Esto permitirá al usuario establecer un nuevo nombre cada vez que lo desee al pulsar el botón:

```
miBoton.onclick = function() {

  estableceNombreUsuario();
}
```

Ahora cuando visites tu sitio por primera vez, este te pedirá tu nombre y te dará un mensaje personalizado de bienvenida. Puedes cambiar cuantas veces quieras el nombre al presionar el botón. Y como un bonus añadido, ya que el nombre se almacena en el localStorage, este permanecerá después de que cierre el sitio, ¡manteniendo ahí el mensaje personalizado cuando abras el sitio la próxima vez!

¿Un nombre de usuario nulo?

Cuando ejecutes el ejemplo y obtengas el cuadro de diálogo que solicita que introduzcas tu nombre de usuario, intenta pulsar el botón Cancelar. Deberías terminar con un título que diga que Mozilla es genial, null. Esto sucede porque, cuando cancelas el mensaje, el valor se establece como null. Null (nulo) es un valor especial en JavaScript que se refiere a la ausencia de un valor.
Además, prueba a dar clic en Aceptar sin introducir un nombre. Deberías terminar con un título que diga que Mozilla es genial, por razones bastante obvias.
Para evitar estos problemas, podrías comprobar que el usuario no ha introducido un nombre en blanco. Actualiza tu función estableceNombreUsuario() a lo siguiente:

```
function estableceNombreUsuario() {
 let miNombre = prompt('Introduzca su nombre.');
 if(!miNombre) {
  estableceNombreUsuario();
 } else {
  localStorage.setItem('nombre', miNombre);
  miTitulo.innerHTML = 'Mozilla is genial, ' + miNombre;
 }
}
```

En el lenguaje humano, esto significa que si miNombre no tiene ningún valor,
ejecute estableceNombreUsuario() de nuevo desde el principio. Si tiene un valor (si la afirmación anterior no es verdadera), entonces almacene el valor en localStorage y establézcalo como el texto del título.

Lo esencial del C#

Variables, constantes y enumeraciones
El nombre de las variables:

- Empiezan por una letra.
- Pueden estar compuestas por caracteres alfanumércios o el carácter subrayado "_".
- Se distinguen mayúsculas de minúsculas
- No se pueden usar las palabras reservadas del lenguaje (if,switch, void, …) a no se que se utilice el caracter arroba "**@**" precediendo al la variable. Por ejemplo.: **@if** sí podría ser una variable.

Tipos numéricos enteros:

- Con signo
  - **sbyte** (8 bits)
  - **short** (16 bits)
  - **int** (32 bits)
  - **long** (64 bits)
- Sin signo
  - **byte** (8 bits)
  - **ushort** (16 bits)
  - **uint** (32 bits)
  - **ulong** (64 bits)

Tipos numéricos decimales (todos son con signo).

- **float** (4 bytes = 32 bits)
- **double** (8 bytes = 64 bits)
- **decimal** (16 bytes = 128 bits)

Tipos de carácter:

- **char** (2 bytes = 16 bits): código unicode donde se pueden representar hasta 256 caracteres, de los cuales los primeros 128 son los mismos que el juego de caracteres ASCII.
- **String**: es el objeto que utilizamos para representar cadenas.

```
Char Car1 = "A";

String Cad1 = "AulaWeb.com.es";
```

Secuencias de escape:

\' = comillas simples	\a = alerta	\r = retorno de carro
\" = comillas dobles	\b = backspace	\t = tabulación horizontal

\\ = barra invertida	\f = salto de página	\v = tabulación vertical
\0 = carácter nulo	\n= salto de línea	

Otra alternativa a la utilización de algunas secuencias de escape la encontramos en el operador “**@**“, que vendría a ser algo parecido a instrucción <pre> de html.

Por ejemplo:

```
cadena = "¿Qué es lo que dice?\rÉl dice \"hola\"";

cadena = @"¿Qué es lo que dice?

Él dice ""hola""";
```

Tipo Boolean:

- **true**
- **false**

```
Boolean EsCierto = true;

Boolean Esfalso = false;
```

Tipo Object:

En una variable tipo Object podemos almacenar cualquier cosa. en realidad la variable almacena la dirección de memoria donde se encuentra el valor de la variable.

Tipo dymanic:

A veces puede ocurrir que sólo podemos conocer el tipo de una variable en tiempo de ejecución y no cuando estamos programando o diseñando la aplicación, para este tipo de casos tenemos la palabra reservada **dynamic** para utilizar junto con la variable afectada y la cual no sabemos de que tipo va a ser.

```
public static dynamic operacion (dynamic op1, dynamic op2) {
    return op1 + op2;
}
```

A esta función “operación” la podríamos invocar por ejemplo:

```
int i1;
int i2;
i1 = 2;
i2 = 3;
Console.WriteLine(operacion(i1,i2));
String s1;
String s2;
s1 = "2";
s2 = "3";
Console.WriteLine(operacion(s1,s2));
Cliente c1;
c1 = new Cliente();
Cliente c2;
c2 = new Cliente();
Console.WriteLine(operación(c1,c2));
```

Las dos primeras llamadas se ejecutarían sin problemas, la primera sumando y la segunda concatenando, la tercera saltaría una excepción ya que la operación “+” no es aplicable al tipo de dato Objeto directamente.

Los tipos Nullables.

para asignar a una variable el valor **Null** (nulo) sólo tenemos que utilizar el carácter “**?**” después del tipo de variable. Por ejemplo podemos hacer la recuperación de ciertos datos de una base de datos y cuando recuperamos esos datos nos damos cuenta que no tienen ningún valor, por ejemplo campos booleanos, enteros. Para estos casos puede ser interesante no asignar ningún valor a la variable, o mejor dicho asignar el valor **Null** a dicha variable. Luego para saber si una variable tiene valor podemos utilizar la función **Hasvalue**. Si intentáramos utilizar el valor de una variable que no tiene valor (null) nos saltaría una excepción, por lo que a las variables que permitimos tener valor nulo siempre deremos tener la precaución de comprobar antes si tienen algún valor asignado.

```
int? CodigoPostal;

float? SueldoBase;

...

CodigoPostal = 46022;

if (CodigoPostal.HasValue) {

    Console.writeLine(CodigoPostal.Value); }

else {

    Console.WriteLine("Código Postal vacío"); }

...

//podemos volver a asignar el valor nulo

CodigoPostal = null;
```

Uso de las variables booleanas con los operadores lógicos.

B1	B2	B1 & B2	B1 | B2
null	null	null	null
null	true	null	true
null	false	false	null
true	null	null	true
true	true	true	true
true	false	false	true
false	null	false	null
false	true	false	true
false	false	false	false

Declaración de variables

Sintaxis general de declaración de una variable (los parámetros entre corchetes son opcionales).

```
Tipo nombreVariable[=valor inicial][,nombreVariable2]
```

En caso de que se omita el valor inicial, la variable será inicializada a **cero** si corresponde a un tipo **numérico**; a una cadena de **carácter vacío** si es del tipo **String**; al valor **null** si es del tipo **Object**, y a **false** si es del tipo **bool**.
Estas reglas no se aplican a las variables declaradas en el interior de una función que deben ser inicializadas antes de poder utilizarse, el siguiente ejemplo daría un error por no haberle asignado ningún valor a la variable.

```
public void test() {

  int i;

  Console.writeLine(i); }
```

Inferencia de tipo

El compilador puede determinar el tipo que se ha de utilizar para una variable local. La inferencia sólo sirve para variables declaradas en una función (variables locales).

```
var apellido = "Pedraza";
```

Las constantes

Valores numéricos o cadenas que no serán modificados durante el funcionamiento de la aplicación.

```
const int ValorMax = 100;

const string Mensaje="Muy grande";

...

If (resultado>Valormax)

  Console.writeLine(Mensaje);

...
```

Podemos también calcular el valor de una constante a partir de otra constante.

```
public const int Total = 100;

public const int Semi = Total / 2;
```

Las enumeraciones

```
enum DiasSemana {Lunes, Martes, Miércoles, Jueves, Viernes, Sábado, Domingo};

// Que sería equivalente a ....

const int Lunes=0;

const int Martes=1;

const int Miércoles=2;

const int Jueves=3;

const int Viernes=4;

const int Sábado=5;

const int Domingo=6;

// Si queremos interrumpir la secuencia de incremento automático

enum DiasSemana2 {Lunes=10, Martes=20, Miércoles=30, Jueves=40, Viernes=50, Sábado=60, Domingo=70};

// Los valores de las enumeraciones siempre son números enteros.

//Podemos utilizar la enumeración como un tipo de variable

DiasSemana2 VarDia;

VarDia = DiasSemana.Lunes;

Console.WriteLine(VarDia);
```

Estructuras de control

- **if**

```
if (condición) { ... }

if (condicion) { ... } else { ... }
```

- **switch**

```
switch (variable) {
    case valor1:  ...
                break;
    case valor2:  ...
                break;
    ...
    deafault:     ...
                break;
}
```

while

```
while (condición) {
   ...
```

```
}
```

❒ do … while

```
do { ... } while (condición);
```

❒ for

```
for (inicialización del contador; condición; instrucción de iteración) { ... }



//Ejemplo

int k1;

for (k1=1; k1 < 10; k1++) {

  for (int k2=1; k2 < 10; k2++) {

    Console.Write(k1 * k2 + "\t"); }

  Console.WriteLine();

}
```

❒ foreach

```
foreach (elemento en matriz) { ... }



//Comparamos for con foreach

string[] matriz={"rojo","verde","azul","blanco"};
```

```
int contador;

for (contador=0; contador < matriz.Length; contador++) {

  Console.WriteLine(matriz[contador]);

}



foreach (string s in matriz) {

  Console.WriteLine(s);

}
```

- **using**

  Esta destinada a simplificar el desarrollo. Permite que en un bloque de código podamos utilizar un recurso externo, en el ejemplo, estamos utilizando los recursos que se encuentran en System.IO.StreamReader.

```
using (System.IO.StreamReader sr = new System.IO.StreamReader(@"C:\Users\Public\Documents\
test.txt"))

{

    string s = null;

    while((s = sr.ReadLine()) != null)

    {

        Console.WriteLine(s);

    }

}
```

Procedimientos y funciones

Todas las instrucciones deben estar incluidas en un procedimiento o función, a las que llamaremos mediante su identificador. A estas funciones y procedimientos podemos pasarles parámetros.

En **C#** tenemos 4 tipos:

1.Los procedimientos que ejecutan un código a petición sin devolver ningún resultado.
2.Las funciones que ejecutan un código y devuelven el resultado al código que las llamó.
3.Los procedimientos de propiedades que permiten manejar las propiedades de los objetos creados.
4.Los procedimientos de operador utilizados para modificar el funcionamiento de un operador cuando se aplica a una clase o una estructura.

Procedimiento

La visibilidad de un procedimiento viene determinada por la declaración ***private***, ***public*** o ***internal***. Por defecto si no se indica nada se entiende que es public.

```
void VerResultado() {

  Console.WriteLine("¡¡¡Ganador!!!");

}
```

Función

La función devuelve un resultado al código invocante. La ejecución de ***return*** provoca la salida de la función.

```
int calculo () {

  ...

  instrucciones

  ...

  return resultado;

}
```

Procedimiento de propiedades

Estos procedimientos se llaman “encapsuladores” ya que el valor de la propiedad se encapsula. Se utilizarán cuando queramos modificar y/o recuperar un valor (Set / Get).

```
public tipoDeLa Propiedad nombrePropiedad {
  get {
    ...
    //código que se ejecuta cuando se lee la propiedad
    ...
    return variable;
  }
  set {
    ...
    //código que se ejecuta durante la asignación de una propiedad
    //existe una variable que se declara implícitamente y que contiene
    //el valor que se debe asignar a la propiedad
    ...
    variable = value;
    ...
  }
}
```

Si una propiedad es de sólo lectura o sólo escritura, se eliminará el bloque **set** y/o **get** correspondiente. También podemos implementar automáticamente la encapsulación cuando no haya tratamiento alguno de la siguiente manera.

```
public int tasa { get; set; }
```

Procedimiento de operador

Permite la redifinicón de un operador estándar del lenguaje para utilizarlo en tipo personalizados (clase o estructura).

```
struct Cliente {
  public int codigo;
  public string apellido;
  public string nombre;
}
  Cliente c1, c2, c3;
  c1.codigo = 200;
  c1.nombre = "Juanjo";
  c1.apellido = "Pedraza";

  c2.codigo = 125;
  c2.nombre = "Perico";
  c2.apellido = "Palotes";
```

```
  c3 = c1 + c2;
  //Aquí el compilador daría error porque no se pueden aplicar el operando al tipo.
}
```

Para que el código anterior funcione se podría hacer esto:

```
struct Cliente {
  public int codigo;
  public string apellido;
  public string nombre;
  public static Cliente operator + (Cliente cl1, Cliente cl2) {
    Cliente c;
    c.codigo = cl1.codigo + cl2.codigo;
    c.apellido = cl1.apellido + cl2.apellido;
    c.nombre = cl1.nombre + cla2.nombre;
    return c;
  }
}
```

Argumentos de los procedimientos y funciones

Por valor:

```
public static double CalculoNETO (double Pbruto, double Tasa) {
```

```
  return Pbruto * (1+(Tasa/100));

}

...

double PrecioBruto = 100;

double PrecioNeto;

PrecioNeto = TestEstructura.CalculoNETO(PrecioBruto, 5.5);

Console.WriteLine(PrecioNeto);
```

En este caso PrecioBruto se está pasando por valor ya que de forma predeterminada los valores enteros, números flotantes, decimales, booleanos y estructuras definidas por el usuario se pasan por valor. Los demás tipos siempre se pasan por referencia, es decir se le pasa la dirección de memoria a la que apunta la variable de forma que el procedimiento o función pueden manipular directamente el valor de esa variable.

Por referencia:

Podemos forzar el paso por referencia de una variable que por defecto se pasa como valor. para ello utilizaremos la palabra reservada "**ref**" o "**out**", la etiqueta "**ref**" se debe utilizar tanto en la lista de parámetros de la función o procedimiento, como en la propia llamada a la función o procedimiento y además debe ser inicializada, por el contrario, la etiqueta "**out**" funciona de igual manera pero sin la exigencia de inicializar la variable.

```
public static double CalculoNETO (double Pbruto, double Tasa, ref double iva) {

  iva = Pbruto * (Tasa/100);

  return Pbruto+iva; }
```

```
...
double PrecioBruto = 100;
double PrecioNeto;
double importeIva=0;
PrecioNeto = TestEstructura.CalculoNETO(PrecioBruto, 5.5, ref importeIVA);
Console.WriteLine("Precio neto: {0}", PrecioNeto);
Console.WriteLine("Importe iva: {0}", importeIva);
```

Ejemplo de procedimiento con número de parámetros indeterminado.

```
public staic double media(params double[] notas) {
  double suma=0;
  foreach (double nota in notas) {
    suma = suma + nota;
  }
  return suma /notas.Length;
}
```

Y alguna de las llamadas a esta función “media” serían:

```
Resultado = media(1,4,67,233);
Resultado = media(23,33);
```

Parámetros opcionales

Se puede indicar que un parámetro es opcional asignándole un valor, pero con la precaución de asignar también a todos los parámetros restantes un valor ya que al declarar un parámetro como opcional, el resto de parámetros también deberían serlo.

```
double calculoNeto(double Pbruto, double Tasa = 21);
```

La siguiente declaración estaría mal ya que el último parámetro no sería opcional:

```
double calculoNeto(double Pbruto, double Tasa = 21; String divisa);
```

Lo correcto sería

```
double calculoNeto(double Pbruto, double Tasa = 21; String divisa="€");
```

Por otro lado en la llamada a la función si se especifica un parámetro opcional, todos los anteriores también se deberían definir:

```
calculoNeto(10);

calculoNeto(10,5.5);

calculoNeto(10,5.5,"$");
```

En cambio esto daría error:

```
calculoNeto(10, ,"$");
```

Parámetros nominados:

Podemos hacer la llamada a la función sin tener en cuenta el orden siempre que nominemos los parámetros. Por ejemplo:

```
calculoNeto(divisa:"$",Pbruto:250);
```

incluso podemos combinar los parámetros por posición y por nominación

```
calculoNeto(10, divisa: "$");
```

pero hay que seguir la regla de que un parámetro nominado sólo puede ser utilizado después de los parámetros por posición, el siguiente ejemplo fallaría:

```
//daría error

calculoNeto(10,Tasa: 5.5 ,"$");
```

Conclusión

En la actualidad nos damos cuenta de que hablar de tecnologías web engloba muchos campos y que sus áreas son diversas y complejas, desde solo consultar información con un navegador ya sea Firefox, safari, opera, chrome, buscar una imagen, consultar sitos de empresas, hacer compras electrónicas, jugar, conocer lugares en 3D, así como consultar datos estadísticos, bases de datos, sitios de gobierno, etc., etc... Las tecnologías juegan un papel muy importante en la web, debido a estas herramientas podemos acceder al internet, que no es solo sentarte frente a un ordenador y abrir el internet sino que hay un gran trabajo detrás para hacer posible este tipo de tecnología, donde miles de personas trabajan actualizando datos, creando los método para mantener todos los enlaces y las comunicaciones.

Referencias

- Tutorial de C#
- (2020).
- Instituto tecnológico de Celaya.
- 2005
- Fundamentos de JavaScript