

DESARROLLO WEB EN ENTORNO SERVIDOR

CAPÍTULO 6:

**Programación con código embebido en el servidor
Funciones y procedimientos.**

Subprogramación

- Un subprograma es un fragmento de código, creado por el usuario, que tiene una funcionalidad específica. Esto permite que el código sea modular y lo podamos reutilizar.
- Tipos:
 - Funciones: subprogramas que devuelven un valor como resultado de su ejecución.
 - Procedimientos: son aquellos que se ejecutan **sin devolver ningún tipo de valor**.

Ventajas de la subprogramación

- Empaquetar tu código en funciones presenta cuatro ventajas importantes:
 1. Reduce la duplicación dentro de un programa, al permitirte agrupar rutinas de uso común en un solo componente.
 2. Una función se crea una vez, pero se utiliza en muchas ocasiones.
 3. Se facilitan la depuración y la prueba del programa cuando éste se encuentra subdividido en funciones, y también se facilita el rastreo de la fuente de errores
 4. Trabajar con funciones alienta el pensamiento abstracto

Funciones y procedimientos

Crear e invocar funciones

- Existen tres componentes para toda función.
 - a) **Argumentos**, que funcionan como los datos de entrada para la función
 - b) **Valores de retorno**, que son los datos de salida presentados por la función.
 - c) El **cuerpo de la función**, que contiene el código que procesa los datos de entrada para transformarlos en datos de salida.

Funciones y procedimientos

Crear e invocar funciones

- Observa éste ejemplo

```
<?php
// definición de función
// muestra el nombre del día de la semana
function diaDeHoy(){
echo "Hoy es " . date('l', mktime());
}
// invocación o llamada de la función
diaDeHoy();
?>
```


Funciones y procedimientos

Crear e invocar funciones

- Para definir una función se emplea:
 - La palabra clave **function**, seguida por el nombre de la función y una lista de argumentos (opcionales) entre paréntesis.
 - Las llaves ({ }) encierran el cuerpo principal de la función, el cual puede contener cualquier código PHP.
 - En PHP invocar funciones es tan sencillo como invocarlas por su nombre (y pasar los argumentos opcionales en caso necesario).

Funciones y procedimientos

Diferencia

- **Sintaxis Función:**

- **PHP:**

```
function nombre($arg1,$arg2,...) {  
    instrucciones;  
    return $valorDevuelto;  
}
```

- **Sintaxis Procedimiento**

- **PHP:**

```
function nombre($arg1,$arg2,...) {  
    instrucciones;  
}
```

Es decir, las funciones devuelven un valor al bloque de programación y el procedimiento no.

Funciones y procedimientos

Crear e invocar funciones

- Nota:
 - En php 4 y posteriores se puede invocar una función antes de definirla, aunque la correspondiente definición de función aparezca más adelante en el programa.
 - Esta particularidad es muy útil para los programadores que prefieren colocar sus definiciones de función al final del script PHP, en lugar de hacerlo al principio, para mejorar la legibilidad.

Funciones y procedimientos

Utilizar argumentos y valores de retorno

- Los argumentos en las funciones aportan los valores de entrada en tiempo de ejecución que necesita la función para completar su ejecución.
- La idea es que como los datos de entrada destinados a la función cambiarán cada vez que ésta sea invocada, los datos de salida siempre serán diferentes.

Funciones y procedimientos

Utilizar argumentos y valores de retorno

- Ejemplo: Realiza éste ejemplo, es el Ejercicio 41 de tu colección.

```
<?php
// definición de funciones
// calcula el perímetro de un rectángulo
//  $p = 2 * (l+a)$ 
function obtenPerimetro($largo, $ancho){
    $perimetro = 2 * ($largo + $ancho);
    echo "El perímetro de un rectángulo con $largo unidades de largo y
    $ancho unidades de ancho es igual a $perimetro unidades";
}
// invocación de la función
// con argumentos
obtenPerimetro(4,2);
?>
```

- También es posible combinar la función con matrices para que devuelva varios valores.

Funciones y procedimientos

Valores por defecto

Por mera conveniencia es posible definir valores por defecto en los argumentos por si no se facilite alguno.

- **Ejemplo:**

```
<?php
```

```
// define una función que genera una dirección de correo electrónico a partir de los valores proporcionados
```

```
function construyeDireccion($nombreusuario, $dominio = 'midominio.info') {  
    return $nombreusuario . '@' . $dominio;  
}
```

```
// invoca la función sin el argumento $dominio, obtendremos ' Mi dirección de correo electrónico es  
juan@midominio.info'
```

```
echo 'Mi dirección de correo electrónico es ' . construyeDireccion('juan');
```

```
//Si facilitamos ambos argumentos el valor por defecto no entra en juego y obtendremos en el  
siguiente ejemplo' Mi dirección de correo electrónico es diana@dominiobueno.net'
```

```
echo 'Mi dirección de correo electrónico es ' . construyeDireccion ('diana', 'dominiobueno.net');
```

```
?>
```

Funciones y procedimientos

Argumentos dinámicos.

- PHP también te permite definir funciones con las llamadas *listas de argumentos de longitud variable*, donde la cantidad de argumentos puede variar de acuerdo con la invocación de la función. Requiere matrices.
- **Ésta forma está indicada cuando el número de argumentos que recibe la función no es conocido o no es siempre el mismo.**

Funciones y procedimientos

Argumentos dinámicos

- **Ejemplo:** Función que suma los valores que recibe y calcula su promedio. (Recuerda, el promedio se calcula sumando todos los valores y dividiendo por el número de valores aportado)

```
<?php
function calPromedio(){
    $args = func_get_args();//func array captura valores con los arg de la función
    $cuenta = func_num_args();//obtiene el número de argumentos
    $suma = array_sum($args);//funcion array que suma los valores
        $prom = $suma / $cuenta;
        return $prom;
}

// invoca la función con 3 argumentos, datos de salida: 6
echo CalPromedio(3,6,9);

// invoca la función con 8 argumentos, datos de salida: 150
echo CalPromedio(100,200,100,300,50,150,250,50);

?>
```


Funciones y procedimientos

Utilizar variables en funciones

- Sabemos que por defecto, las variables utilizadas dentro de una función son locales
- Sin embargo, en algunas ocasiones será necesario “importar” una variable del programa principal a una función o viceversa. Para tales casos, PHP cuenta con la palabra clave **global**: cuando se aplica a una variable dentro de una función, esta palabra clave la convierte en una variable global, es decir, la hace visible tanto dentro como fuera de la función que la contiene.

Gracias por tu interés

Fin de la presentación