RESUMEN SUBPROGRAMAS. UT3

1. Concepto subprograma.

   Fragmento de un programa. Fragmentando un programa se consigue un código modular que facilita la codificación y su posterior mantenimiento, además de favorecer su reutilización.

   En muchos lenguajes se distingue entre funciones y procedimientos; en PHP no, sólo se utiliza el termino función.

2. Sintaxis.
   a. Descripción.

      function nombre_funcion([parámetros o argumentos]): [tipo resultado]

   b. Definición.

      function nombre_funcion([parámetros o argumentos])
      {
      Sentencias;
      [return;]
      }

   c. Invocación.

      nombre_función ([parámetros o argumentos]);

3. Paso de parámetros.

Normalmente, se establece una correspondencia entre los parámetros establecidos en la llamada a una función y los especificados en la cabecera de su definición, aunque PHP permite incluir una asignación de valor a un parámetro en la cabecera de la definición de una función y con ello, que el número de parámetros incluidos en la llamada a la función, sea un número de parámetros variable.

Tipos:

   a. Por valor, y se pasa una copia del parámetro.
   b. Por referencia y se pasa la dirección de memoria del parámetro (se antepone al parámetro o argumento el símbolo &).

4. Resultados obtenidos (accesibles posteriormente desde el programa o módulo que invocó a la función).
   a. A través de la sentencia return; el tipo de resultado indicado en la sintaxis de la declaración, será compatible con el tipo de dato especificado en está sentencia.
   b. A través del paso de parámetros por referencia

5. Ámbito de las variables:

Contexto dentro del que la variable está definida; es decir, la parte de la aplicación en la que puede ser accedida. Una variable es visible allí donde por primera vez, toma valor.

Las variables internas a una función, por defecto, sólo pueden ser utilizadas dentro de dicha función, reservándose un espacio en la memoria para ellas, luego fuera de ella, no tienen validez. Se dice que son variables ***locales*.**

Esto se puede modificar:

- pasando como parámetro la variable (siendo necesario el paso por referencia para ser modificado su valor)
- con el modificador de variable ***global***, de este modo una variable externa a una función puede ser utiliza dentro de ella (no es un uso recomendable).

- con el modificar *static* con el fin de mantener el valor de una variable cuando se invoca varias veces a la función y se quiere conservar su valor (también se puede hacer utilizando el paso de parámetros por referencia)

```php
// Caso 1

$var1=1;
function ambito1()
{
 $var2=$var1;
 echo "Pasando por ambito1(): <br>";
 echo var_dump($var2)."<br>"; //valor null debido a que $var1 no existe en la función
}
ambito1();
```

```php
//Caso 2

$var1=1;

function ambito2()
{
 global $var1;
 $var2=$var1;
 echo "Pasando por ámbito2(): <br>";
 echo $var2."<br>"; // el valor que toma $var2 es 1.
}
ambito2();
```

```php
//Caso 3

$var1=1;
function ambito3()
{
 global $var1;
 $var1++;
 echo "Pasando por ámbito3(): <br>";
 echo $var1."<br>";
}
 ambito3();
 echo $var1."<br>"; // el valor que toma $var1 es 2.
```

```php
// Caso 4
$var1=1;
function ambito4()
{
 global $var1;
 static $var3=1; // una variable estática debe inicializarse a la vez que se declara
                 // como static.
                  // solo se inicializa la primera vez que se llama a la función
                  // el resto de las veces se utiliza el valor actualizado
 $var2=$var1+$var3;
 $var3+=10;
 echo "Pasando por ambito4():<br>";
 echo $var2." ".$var3."<br>";
 }
ambito4();
ambito4();
```

```php
//Caso 5
// uso de paso de paramétros y return en una función

$var1=1;
function ambito5($p)
{
 $p++;
 return $p;
}
 echo "Pasando por ambito5():<br>";
 echo ambito5($var1)."<br>"; // el valor que toma $var1 es 2.
```

```php
//Caso 6
// uso de paso de parámetros por referencia

$var1=1;
function ambito6(&$p)
{
 $p++;
}
 ambito6($var1);
 echo "Pasando por ambito6():<br>";
 echo $var1."<br>"; // el valor que toma $var1 es 2.
```

En PHP existes variables ***superglobales*** y que son accesibles en cualquier entorno, es decir, en cualquier archivo de una aplicación web. Son variables tipo array que contienen un conjunto de valores. Estas variables ($_GLOBALS, $_SERVER, $_REQUEST, $_POST, $_GET, $_FILES, $_ENV, $_COOKIE y $_SESSION) serán muy útiles en próximos apartados y UT de este módulo.

6. Clasificación:
    a. Predefinidas en PHP
    b. Definidas por el usuario.

7. Funciones predefinidas en PHP: (https://www.php.net/manual/es/indexes.functions.php)

Relacionadas con el tipo de dato de una variable (UT2)

- gettype(), devuelve un string indicando el tipo del dato pasado como parámetro
- is_int(), is_bool(), is_float(),…., devuelve true o false dependiendo del tipo del dato pasado como parámetro
- settype(), convierte una variable al tipo especificado, devolviendo true si la conversión ha sido posible.
- var_dump()
- isset ()
- unset(), admite varias variables.
- empty(), cadena vacía.

Funciones matemáticas:

- rand()
- round()
- pow()
- sqrt()
- …

Manejo de arrays (UT3 libro de texto)

- Array(): inicializar un array
- Unset(): destruir una component e o el array completo
- Count(): contabilizar componentes
- Is_array(): comprobar si es una variable array
- In_array():buscar una componente en el array
- Array_search():buscar una componente en array
- Sort():ordenar un array de menor a mayor
- Rsort():ordenar un array de mayor a menor
- Array_merge():unir dos arrays
- Array_pop():eliminar el último elemento de un array
- Array_push():añadir un elemento al final
- List(): listar los componentes de un array
- r_print(): visualiza un array

Para el manejo de string (UT3 libro de texto)

- echo()
- print()
- printf()
- sprintf()
- str_split()
- ctype()

- explode()
- implode()
- Ltrim()
- …

Seleccionadas con las fechas y horas.

En PHP no existe un tipo de dato específico para las fechas y las horas; se utiliza el tipo entero. Sin embargo hay múltiples funciones para su tratamiento

- date (formato,[int $fechahora])), permite dar formato a una fecha; si no se incluye como parámetro se toma la generada por el sistema.

Carácter	Resultado
d	día del mes con dos dígitos.
j	día del mes con uno o dos dígitos (sin ceros iniciales).
z	día del año, comenzando por el cero (0 = 1 de enero).
N	día de la semana (1 = lunes, ..., 7 = domingo).
w	día de la semana (0 = domingo, ..., 6 = sábado).
l	texto del día de la semana, en inglés (Monday, ..., Sunday).
D	texto del día de la semana, solo tres letras, en inglés (Mon, ..., Sun).
W	número de la semana del año.
m	número del mes con dos dígitos.
n	número del mes con uno o dos dígitos (sin ceros iniciales).
t	número de días que tiene el mes.
F	texto del día del mes, en inglés (January, ..., December).
M	texto del día del mes, solo tres letras, en inglés (Jan, ..., Dec).
Y	número del año.
y	dos últimos dígitos del número del año.
L	1 si el año es bisiesto, 0 si no lo es.
h	formato de 12 horas, siempre con dos dígitos.
H	formato de 24 horas, siempre con dos dígitos.
g	formato de 12 horas, con uno o dos dígitos (sin ceros iniciales).
G	formato de 24 horas, con uno o dos dígitos (sin ceros iniciales).
i	minutos, siempre con dos dígitos.
s	segundos, siempre con dos dígitos.
u	microsegundos.
a	am o pm, en minúsculas.
A	AM o PM, en mayúsculas.
r	fecha entera con formato RFC 2822.

- date_default_timezone_set(), para establecer la zona horaria
- getdate(), devuelve un array con información sobre la fecha y la hora actual.

8. Creación de librerías.

   Es necesario utilizar la sentencia *include* o *require* y especificar el nombre del archivo donde están las funciones. La diferencia entre ambas está en que con include si el archivo no existe, sí son interpretadas y ejecutadas el resto de las sentencias que haya a continuación de esta sentencia y con require, no se ejecutan más sentencias.