Ud3. Programación. PHP.

RA3. Escribe bloques de sentencias embebidos en lenguajes de marcas, seleccionando y utilizando las estructuras de programación. (10%)

Criterios de evaluación.

a) Se han utilizado mecanismos de decisión en la creación de bloques de sentencias.

b) Se han utilizado bucles y se ha verificado su funcionamiento.

c) Se han utilizado «arrays» para almacenar y recuperar conjuntos de datos.

d) Se han creado y utilizado funciones.

e) Se han utilizado formularios Web para interactuar con el usuario del navegador Web.

f) Se han empleado métodos para recuperar la información introducida en el formulario.

g) Se han añadido comentarios al código.

Contenido.

1. Introducción.
2. Sentencias de control.
3. Funciones.
4. Arrays.
5. Gestión de formularios.

Introducción.

La programación estructurada es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa de computadora recurriendo únicamente a subrutinas y tres estructuras básicas: secuencia, selección (***if*** *y* ***switch***) e iteración (bucles ***for*** *y* ***while***).

El teorema del programa estructurado es un resultado en la teoría de lenguajes de programación. Establece que toda función computable puede ser implementada en un lenguaje de programación que combine sólo tres estructuras lógicas. Esas tres formas (también llamadas estructuras de control) específicamente son:

- Secuencia: Ejecución de una instrucción tras otra.
- Selección: Ejecución de una de dos instrucciones (o conjuntos), según el valor de una expresión lógica.
- Iteración: Ejecución de una instrucción (o conjunto) mientras una expresión lógica sea verdadera. Esta estructura lógica también se conoce como ciclo o bucle.

Estructuras de control.

Bloque de código: Conjunto de instrucciones agrupado entre llaves: { }. Un bloque de una única instrucción no necesita ir entre llaves.

PHP ofrece una sintaxis alternativa para alguna de sus estructuras de control: **if, while, for, switch**. En cada caso, la forma básica de la sintaxis alternativa es cambiar la llave de apertura { por dos puntos : y la de cierre } por **endif;, endwhile;, endfor;, o endswitch;.**

```
if ($a == 5):
    print "a es igual a 5";
    print "...";
elseif ($a == 6):
    print "a es igual a 6";
    print "!!!";
else:
    print "a no es ni 5 ni 6";
endif;
```

Estructuras de control de selección.

Las estructuras de control de selección se utilizan para tomar decisiones en función de ciertas condiciones. Las estructuras de control de selección más comunes son: `if` y `switch`.

`if`

Se utiliza para ejecutar un bloque de código si una condición es verdadera. Si la condición no se cumple, el bloque de código no se ejecuta.

```
if (expLogica) {

        // Código a ejecutar si la condición es verdadera

}
```

`if … else`

En este caso se utiliza **else** para ejecutar un bloque de código enn el único caso en el que la condición que acompaña a **if** sea falsa.

```
if (expLogica) {
    // Código a ejecutar si la condición es verdadera
}
else {
    // Código a ejecutar si la condición es falsa
}
```

`if … else if`

Se puede utilizar **else if** después de un **if** para verificar múltiples condiciones en secuencia. Solo se ejecutará el bloque de código asociado a la primera condición verdadera.

Es necesario tener en cuenta que **elseif** y **else if** serán considerados exactamente iguales sólamente cuando se utilizan llaves. Al utilizar los dos puntos para definir las condiciones **if/elseif**, no debe separarse **else if** en dos palabras o PHP fallará con un error del interprete.

```
if (expLogica1) {
   // Código a ejecutar si la condición1 es verdadera
}
else if (expLogica2) {
   // Código a ejecutar si la condición2 es verdadera
}
```

Es posible utilizar **else** para ejecutar un bloque en el caso de que ninguna de las condiciones sea verdadera.

switch

La estructura **switch** se utiliza para evaluar una expresión y ejecutar diferentes bloques de código según el valor de la expresión. Cada bloque de código se asocia a un caso (case) específico. Es similar a una serie de sentencias **if**. 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 es igual.

```
switch (expresión) {
   case valor1:
      // Código a ejecutar si la expresión coincide con valor1
      break;
   case valor2:
      // Código a ejecutar si la expresión coincide con valor2
      break;
   default:
      // Código a ejecutar si la expresión no coincide con ningún caso
}
```

La sentencia **switch** comienza a ejecutar las instrucciones cuando se encuentra una cláusula case con un valor que coincide con el valor de la expresión **switch**. PHP ejecutará todas las instrucciones que se encuentre hasta el final del bloque, o hasta encontrar la cláusua **break**. Si no se escribe una sentencia **break** al final de una lista de opciones case, PHP seguirá ejecutando las sentencias del siguiente case.

Estas son las estructuras de control de selección más comunes en PHP. Se utilizan para tomar decisiones en el código y ejecutar diferentes acciones en función de las condiciones que se cumplan. Es posible combinar estas estructuras y anidarlas para manejar situaciones más complejas de toma de decisiones en los programas PHP.

Ejemplo:

```php
$edad = 25;
if ($edad >= 18) {
    echo "Eres mayor de edad.";
}
else {
    echo "Eres menor de edad.";
}
```

Ejemplo:

```php
$nota = 85;
if ($nota >= 90) {
    echo "Tienes una A.";
}
else if ($nota >= 80) {
    echo "Tienes una B.";
}
else if ($nota >= 70) {
    echo "Tienes una C.";
}
else {
    echo "Tienes una D.";
}
```

Ejemplo:

```php
$numero = 7;
if ($numero % 2 == 0) {
    echo "El número es par.";
}
else {
    echo "El número es impar.";
}
```

Ejemplo:

```php
$dia = "Miércoles";
switch ($dia) {
   case "Lunes":
      echo "Hoy es el primer día de la semana.";
      break;
   case "Martes":
   case "Miércoles":
   case "Jueves":
   case "Viernes":
      echo "Hoy es un día laborable.";
      break;
   case "Sábado":
   case "Domingo":
      echo "Hoy es fin de semana.";
      break;
   default:
      echo "Día no válido.";
}
```

Ejemplo:

```php
$i = 1;
switch ($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");
}
```

- ¿Cuál es el resultado de la ejecución del script cuando $i tome el valor 1?

Aquí sí $i es igual a 0, PHP ejecutará todas las sentencias print. Si $i es igual a 1, PHP ejecutará 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'.

Es importante no olvidar las sentencias ***break***.

En una sentencia ***switch***, la condición se evalúa sólo una vez y el resultado se compara con el valor de cada cláusula case. En una sentencia ***elseif***, la condición se

evalúa otra vez. Cuando la condición es compleja suele ser más eficiente utilizar ***switch***.

Los bloques de código de las cláusulas ***case*** pueden estar vacíos, en cuyo caso, el flujo del programa continúa en el bloque de código del siguiente ***case***.

Ejemplo:

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

- ¿Cuál es el resultado de la ejecución del script cuando $i tome el valor 0?

La cláusula **default** permite incluir un bloque de código que se ejecutará cuando el valor de la expresión **switch** no coincida con ninguno de los valores que acompañan al resto de cláusulas case.

Ejemplo:

```php
<?php
    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';
}
```

- ¿Cuál es el resultado de la ejecución del script cuando $i tome el valor 3?

Estandar de codificación:

- Las sentencias de control basadas en **if** y **elseif** deben tener un único espacio en blanco antes del paréntesis inicial y un único espacio en blanco después del paréntesis final.
- Los operadores dentro de la condición deben separarse con espacios. Es recomendable usar paréntesis internos para mejorar la agrupación de las expresiones condicionales largas.
- La llave de inicio { debe estar en la misma línea que la condición, mientras que la de cierre } debe estar sola. El código dentro de las llaves debe estar indentado.

  Ejemplo:

```php
<?php
    if ($a != 2) {
        $a = 2;
    }
```

- Si el largo de la condición es mayor al largo máximo de la línea, y la condición tiene varias cláusulas, se puede partir en varias líneas.

  Ejemplo:

```php
<?php
    if (($a == $b) &&
        ($b == $c) ||
        (Foo::CONST == $d)) {
        $a = $d;
    }
```

- Toda sentencia **if**, **elseif** o **else** debe usar llaves de apertura { y cierre } sin excepción.
- Se recomienda utilizar **else if** en vez de **elseif**.
- El código dentro de un bloque **switch** y dentro de cada case debe estar indentado.

  Ejemplo:

```php
<?php
   switch ($respuestaPregunta) {
       case 1:
           break;
       case 2:
           break;
       default:
           break;
   }
```

- Nunca se debe omitir la cláusula **default** en una instrucción **switch**.
- Si se omite intencionalmente un **break** dentro de un case, debe escribirse un comentario justificando el motivo.

ACTIVIDADES DE CLASE.

1. Almacena tres números en variables y escribirlos en pantalla de manera ordenada.
2. Carga en variables mes y año e indica el número de días del mes. Utiliza la estructura de control **`switch`**
3. Carga fecha de nacimiento en variables y calcula la edad.
4. Modifica la página inicial realizada, incluyendo una imagen de cabecera en función de la estación del año en la que nos encontremos y un color de fondo en función de la hora del día.
5. Script que muestre una lista de enlaces en función del perfil de usuario:

   Perfil Administrador: Pagina1, Pagina2, Pagina3, Pagina4

   Perfil Usuario: Pagina1, Pagina2

Estructuras de control repetitivas.

Las estructuras de control repetitivas permiten ejecutar un bloque de código varias veces según una condición específica. Las estructuras de control repetitivas más comunes en PHP son **for**, **while**, **do...while** y **foreach**.

`while`

Ejecuta el bloque de código mientras la expresión lógica devuelva como resultado true.

```php
<?php
while (expLogica) {
  // Código a ejecutar mientras la condición sea verdadera
}
```

Conviene recordar como trata PHP los tipos de datos cuando aparecen en expresiones lógicas.

En función del valor inicial de la expresión lógica, puede darse el caso de que el bloque de código no se ejecute nunca.

Es necesario crear en el bloque las condiciones necesarias para garantizar que en algún momento la expresión lógica se evalúe como FALSE y evitar bucles infinitos.

También es posible utilizar la sintaxis alternativa:

```php
<?php
while (expLogica):
  // Código a ejecutar mientras la condición sea verdadera
endwhile
```

Ejemplo:

```php
<?php
    $i = 1;
    while ($i <= 10) {
        print $i++;
    }
```

Ejemplo:

```php
<?php
    $i = 1;
    while ($i <= 10):
        print $i;
        $i++;
    endwhile;
```

do … while

Los bucles **do...while** son muy similares a los bucles ***while***, excepto que las condiciones se comprueban al final de cada iteración en vez de al principio.

```php
<?php

do {
  // Código a ejecutar al menos una vez
} while (expLogica);
```

La principal diferencia frente a los bucles regulares **while** es que los bucles **do...while** garantizan la ejecución de la primera iteración.

Es necesario garantizar que la expresión lógica llegue a evaluarse como false para evitar bucles infinitos.

El bucle **do...while** no admite sintaxis alternativa.

Ejemplo:

```php
<?php

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

- ¿Cuál es el resultado de la ejecución del script?

El bucle se ejecuta exactamente una vez. Después de la primera iteración, la expresión lógica se evalúa como FALSE y la ejecución del bucle finaliza.

for

Por lo general la estructura **for** se utiliza cuando se conoce el número de iteraciones del bucle, aunque en PHP es posible utilizarlo en otras condiciones.

```php
<?php
  for (expInicializacion; expLogica; expActualizacion) {
     // Código a ejecutar en cada iteración
  }
```

La primera expresión de inicialización se evalúa incondicionalmente una vez al principio del bucle. Al comienzo de cada iteración se evalúa la expresión Lógica . Si se evalúa como true, el bucle continúa y se ejecuta el bloque de código. Si se evalúa como false la ejecución del bucle finaliza. Al final de cada iteración se evalúa la expresión de actualización, por lo general una acción de incremento o decremento.

Cada una de las expresiones puede estar vacía. Si la expresión lógica está vacía, es interpretada como true y el bucle se ejecutaría indefinidamente.

Ejemplos.

```php
<?php
    for ($i = 1; $i <= 10; $i++) {
        print $i;
    }
```

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

También es posible utilizar la sintaxis alternativa:

```php
<?php
  for (expInicializacion; expLogica; expActualizacion):
     // Código a ejecutar en cada iteración
  endfor;
  }
```

foreach

Se trata de un bucle especial que estudiaremos con arrays.

Estandar de codificación:

Utilizamos los mismos criterios que en las estructuras condicionales.

ACTIVIDADES DE CLASE.

1. Escribir los números 1 al 10
2. Sumar los 3 primeros números pares.
3. Tablas de multiplicar del 1 al 10. Aplicar estilos en filas/columnas
4. Mostrar paleta de colores. Utilizar una tabla que muestre el color y el valor hexadecimal que le corresponde. Cada celda será un enlace a una página que mostrará un fondo de pantalla con el color seleccionado. ¿Puedes hacerlo con los conocimientos que tienes?
5. Dado el mes y año almacenados en variables, escribir un programa que muestre el calendario mensual correspondiente. Marcar el día actual en verde y los festivos en rojo.

ACTIVIDADES DE CLASE.

1. Escribe un programa que genere e imprima 20 números aleatorios (0-100), mostrando también el mayor, el menor y la media.
2. Escribe un programa que genere e imprima un número aleatorio de 4 cifras, mostrando a continuación cada una de sus cifras en un color diferente.
   Por ejemplo:

   6792

   2
   9
   7
   6

3. Escribe un programa que genere tres números aleatorios correspondientes a la fecha de nacimiento (día, mes, año) de una persona. Debes garantizar que la fecha generada es válida.
   El script mostrará en pantalla la fecha y una imagen con el signo del zodiaco de la persona.

4. Programa que lea un número entero N de 5 cifras y muestre sus cifras igual que en el ejemplo.
   Ejemplo.

   12345

   5
   45
   345
   2345
   12345

Funciones y librerías.

- Funciones predefinidas.
- Funciones de usuario.
- Incorporando ficheros.

Funciones predefinidas

Consulta funciones en php.net

Funciones de usuario.

Una función es un bloque de código que constituye una **unidad funcional** destinada a resolver un problema concreto. Es necesario declarar la función para posteriormente utilizarla mediante una llamada.

Declaración de una función

```
function nombreFuncion([argumentos])[:tipo]
{
    bloque
   [return valor;]
}
```

Los argumentos (o parámetros formales) de la función son una lista de declaraciones de variables separadas por comas.

La instrucción **return** es utiliza por la función para retornar un valor a la llamada.

Llamada a una función.

```
nombreFuncion([parametos]);
```

Los parámetros, llamados actuales, en la llamada a una función son una lista de expresiones que se asignan por posición a los argumentos declarados en la definición de la función. La asignación se hace por orden y no por nombre.

```php
<?php
/**
* Función que escribe un mensaje.
*
* @param string $mensaje a mostrar.
*/
function writeMsg($mensaje)
{
    echo “Hello, “;
    echo $mensaje;
}
writeMsg(“2dwes”); // Llamada a la función
```

Las variables declaradas dentro de una función y los parámetros son locales y por tanto accesibles sólo desde dentro de la función.

Desde una función no es posible acceder a las variables definidas fuera de ella.

$GLOBALS es un array que almacena las variables globales declaradas.

No se recomienda el uso de variables globales.

Variables estáticas.

Son variables que tienen la propiedad de conservar el valor de una llamada a otra función, durante la ejecución del script. De forma predeterminada, las variables locales de una función se restablecen en cada llamada a la función.

static $variable = expresión

Ejemplo:

```php
<?php
    /**
    * Ejemplo de uso de variables estáticas..
    */
    function manejoVariablesEstaticas()
    {
        static $varEstatica= 0;
        $variable=0;

        $varEstatica++;
        $variable++;

        echo $varEstatica;
        echo $variable;
    }
    echo “<p>Llamada 1</p>”;
    manejoVariablesEstaticas();
    echo “<p>Llamada 2</p>”;
    manejoVariablesEstaticas();
    echo “<p>Llamada 3</p>”;
    manejoVariablesEstaticas();
```

Paso de parámetros por valor/referencia.

Hay dos formas de pasar parámetros a una función

- **Por valor**. El cambio de valor en los argumentos declarados en la función no modifica el valor de los parámetros en la llamada.
- **Por referencia**. Cuando los parámetros son pasados por referencia a una función, su valor se establece al valor que tenga su correspondiente argumento en la declaración. El paso por referencia se realiza utilizando el operador de referencia **&**

Ejemplo:

```php
<?php
    /**
    * Paso de parámetros por valor y por referencia.
    * @param int $refX Variable pasada por referencia.
    * @param int $y Variable pasada por valor.
    */
    function pasoPorReferencia(&$refX, $y)
    {
        $refX ++;
        $y ++;
    }

    $var1=5;
    $var2=-5;
    echo $var1;
    echo $var2;
    echo "<br/>";
    pasoPorReferencia($var1, $var2);
    echo $var1;
    echo $var2;
```

Argumentos por defecto

Es posible utilizar parámetros predeterminados en la declaración de la función. Será el valor que toma el argumento cuando no reciba un parámetro en la llamada.

Ejemplo:

```php
?php
    /**
    * Establece valor por defecto.
    * @param string $color.
    */

    function setColor($color='Rojo')
    {
        echo "El color es : $color <br>";
    }

    setColor('Verde');
    setColor(); // Valor por defecto Rojo
```

Manejo de argumentos. Funciones.

PHP incorpora un conjunto de funciones para el manejo de argumentos.

func_num_args: Número de parámetros pasados a la función.

func_get_args: Devuelve lista de parámetros pasados a la función (en un array)

func_get_arg: Devuelve el valor de un parámetro cuyo número se especifica.

Devolución de valores.

El tipo de valor devuelto en una función se puede especificar en la cabecera de la misma.

Para que una función devuelve un valor se utiliza la sentencia **return**

```php
<?php
    /**
    * Función que suma dos números.
    * @param int $x
    * @param int $y
    *
    * @return int $z
    */
    function suma($x,$y): int
    {
        $z=$x+$y;
        return $z;
    }

    echo "5 + 10 = " . suma(5,10) . "<br/>";
    echo "7 + 13 = " . suma(7,13) . "<br/>";
    echo "2 + 4 =  " . suma(2,4);
```

Recursividad

La recursividad es la propiedad que permite a una función llamarse a sí misma. Su importancia se encuentra en la existencia de numerosos problemas cuya solución está basada en algoritmos recursivos.

Son algoritmos muy potentes de fácil implementación.

De manera general los algoritmos recursivos tendrán una parte responsable de la llamada y otra encargada de realizar la ruptura en la sucesión de llamadas.

Ejemplo:

```php
<?php
    /**
    * Función factorial recursivo
    * @param int $n
    *
    * @return int $factorial
    */
    function factorial($n)
    {
        if ($n == 0) {
            return 1;
        }
        else {
            return $n * factorial($n-1);
        }
    }
    echo factorial(5);
```

Función anónima.

Es una función a la que no se le ha asignado ningún nombre en su declaración.

Se puede utilizar como el valor de una variable o como una función de devolución de llamada a otra función.

Ejemplo:

```php
<?php
    /**
    * Función anónima
    * @param string $mensaje
    *
    */

    $saludo = function($mensaje)
    {
        echo "Contenido del mensaje: $mensaje<br/>";
    };
    $saludo("Hola Mundo");
    $saludo("2 DWES");
```

Función variable.

PHP permite almacenar el nombre de una función en una variable y llamar a la función utilizando la variable con la notación ***$variable()***

Ejemplo.

```php
<?php
    /**
    * Función que calcula el producto de dos números
    * @param int $var1
    * @param int $var2
    * @return int
    *
    */
    function producto($var1, $var2)
    {
       return $var1 * $var2;
    }
    /**
    * Función que calcula la suma de dos números
    * @param int $var1
    * @param int $var2
    * @return int
    *
    */
    function suma($var1, $var2)
    {
        return $var1 + $var2;
    }
    /**
    * Función que variable(suma, producto) de dos números
    * @param function $operacion
    * @param int $var1
    * @param int $var2
    *
    * @return int
    *
    */
    function calcular ($operacion, $var1, $var2)
    {
        return $operacion($var1, $var2);
    }
```

```
echo calcular('suma',3,5);
echo '<br/>';
echo calcular('producto',3,5);
```

Type hinting.

Las últimas versiones de PHP permiten la determinación de tipos, lo que permite especificar el tipo de dato que espera un argumento en la declaración de una función. Cuando se llama a la función, PHP comprobará si los argumentos son del tipo especificado, y sino lo son se emitirá un error y la ejecución se detendrá.

Los tipos soportados por esta característica (en PHP 5.5) son: arrays, clases, interfaces y tipos escalares.

El objetivo de la determinación de tipos es organizar mejor el código y mejorar el manejo de los errores.

Incorporando ficheros.

PHP permite incorporar el contenido de un fichero en otro. Para ello disponemos de las siguientes funciones:

include, include_once, require y requiere_once

include, include_once: Devuelven verdad en caso de éxito y falso en caso de error. En este último caso generan un error de nivel E_WARNING que no interrumpe la ejecución del script.

require, require_once: No tienen código de retorno y si la operación no tiene éxito se genera un error faltal que interrumpe la ejecución del script.

Cuando utilizamos las funciones con los sufijos ***_once*** garantizamos que un archivo se incluye una sola vez, aunque se le llame varias veces.

Se suelen utilizar para:

Definiciones estáticas: constantes y definición de funciones. En este caso es recomendable utilizar ***..._once***.

Incluir código PHP o HTML dinámico que se ejecuta efectivamente en el momento de inclusión: Sección HTML o código común en varias páginas (encabezado, pie de página, ...). Se recomienda utilizar **include** o **require**.

Estandar de codificación

- Los nombres de las funciones deben estar de acuerdo a las convenciones de nombrado de Zend Framework.
- La llave “{” debe escribirse en la siguiente línea del nombre de la función (“one true brace”).
- No se permite un espacio entre el nombre de la función y el paréntesis de apertura de los argumentos.
- No se permiten las funciones con un alcance global.
- En caso de que los argumentos sobrepasen el tamaño máximo de la línea, se puede partir la declaración.

- Los nombres de las funciones sólo pueden contener caracteres alfanuméricos. No se permiten los guiones bajos (_) y no se recomiendan los números.
- Los nombres de las funciones deben empezar con una letra minúscula, pero cuando se compone de varias palabras, de la segunda palabra en adelante deben iniciar con letra mayúscula (camel case).
- Los nombres deben ser descriptivos de su propósito y comportamiento.
- No se permite la llamada por referencia a los parámetros.
- El valor de retorno no debe estar indicado entre paréntesis.
- Los valores de los argumentos en una función deben separarse por un espacio después de la coma.
- Al pasar arrays como argumentos de una función se puede utilizar el indicador y se pueden separar en varias líneas para facilitar la legibilidad:

Arrays.

Un array es una estructura de datos que contendrá un conjunto de variables con un nombre común y un índice para referenciarlas individualmente. El índice acompaña al nombre del array entre corchetes, [ ].

Clasificación.

En función del tipo de índice:

- Arrays indexados: Arrays con índice numérico.
- Arrays asociativos: Arrays que utilizan como índice una cadena de caracteres.

En función del número de índices.

- Vectores
- Matrices
- Arrays multidimensionales: Arrays cuyo contenido es uno o más arrays y por tanto necesitan varios índices para el acceso a los valores.

Arrays indexados.

Son arrays que utilizan un índice numérico. El primer índice puede ser cualquier número, pero se recomienda empezar en cero.

Hay dos formas de crear un array indexado:

Utilizando la función **`arrray`**().

```php
<?php
    $cars = array("Volvo", "BMW", "Toyota");
```

Asignando directamente el contenido.

```php
<?php
    $cars[0] = "Volvo";
    $cars[1] = "BMW";
    $cars[2] = "Toyota";
```

Es posible crear nuevos elementos con asignación automática de índices utilizando los corchetes sin índice. Cuando se asigna un valor a una variable array usando corchetes vacíos, el valor se añadirá al final del array.

```php
<?php
    $cars[]='Renault';
    $cars[]='Citroen';
?>
```

Arrays asociativos.

Son arrays que utilizan como índices cadenas de caracteres.

Hay dos maneras de crear un array asociativo:

- Utilizando la función **array**()

```php
<?php
    $age = array("Peter"=>35,
                 "Ben"=>37,
                 "Joe"=>43);
?>
```

- Asignando directamente el contenido:

```php
<?php
    $age['Peter'] = 35;
    $age['Ben'] = 37;
    $age['Joe'] = 43;
?>
```

Trabajando con arrays

Longitud de un array. Función **count ()**

```php
<?php
    $cars = array('Volvo', 'BMW', 'Toyota');
    echo count($cars);
?>
```

Recorrer un array indexado. Es necesario utilizar un bucle.

```php
<?php
    $cars = array("Volvo", "BMW", "Toyota");
    $arrLength = count($cars);
    for($x = 0; $x < $arrLength; $x++) {
        echo $cars[$x];
        echo "<br>";
    }
?>
```

Recorrer un array asociativo. Para recorres y procesar todos los valores de un array asociativo se recomienda utilizar el bucle **foreach**

```php
foreach (expArray as [$clave=>]$valor) {
  // Código que se ejecuta en cada iteracción.
}
```

El bucle recorre el array dado por expArray. En cada iteración, el valor del elemento actual se asigna a $valor y el puntero interno del array avanza una posición. $valor es tratado en el bloque de instrucciones del bucle.

Cuando en la instrucción se incluye $clave, en cada iteración, el valor del índice se asigna a $clave y el valor del elemento actual se asigna a $valor. El puntero interno del array avanza una posición. $clave y $valor son procesados en el bloque de instrucciones del bucle.

Ejemplo:

```php
<?php
    $age = array(
              "Peter"=>35,
              "Ben"=>37,
              "Joe"=>43);
    foreach ($age as $valor) {
        echo $valor;
    }
?>
```

El resultado del código.

1ª Iteración.

$valor<-35

2ª Iteración.

$valor<-37

3ª Iteración.

$valor<-43

```php
<?php
    $age = array(
              "Peter"=>35,
              "Ben"=>37,
              "Joe"=>43);
    foreach ($age as $clave=>$valor) {
        echo $clave . " : " . $valor;
    }
?>
```

El resultado del código.

1ª Iteración.

$clave<-Peter

$valor<-35

2ª Iteración.

$clave<-Ben

$valor<-37

3ª Iteración.

$clave<-Joe

$valor←43

Ejemplo:

```php
<?php
    $array = array(1, 2, 3, 4);
    foreach ($array as $valor) {
        echo $valor;
    }
```

- ¿Cuál es el resultado del código anterior?

Ejemplo:

```php
<?php
    $age = array("Peter"=>35,"Ben"=>37,"Joe"=>43);
    foreach($age as $x=>$x_value) {
       echo "Key=" . $x . ", Value=" . $x_value;
       echo "<br/>";
    }
?>
```

Arrays multidimensionales.

Cada variable de un array puede ser a su vez un array, de esta manera podemos crear arrays de dos, tres o más dimensiones. Por lo general, para manejar un array multidimensional necesitaremos un índice por cada una de las dimensiones del mismo.

La función **array**() se puede anidar para crear arrays multidimensionales.

Ejemplo:

Array indexado bidimensional donde cada elemento es un array asociativo.

```php
<?php
    $cars = array (
                    array("marca"=>"Volvo", "m1"=>100, "m2"=> 96),
                    array("marca"=>"BMW", "m1"=>60, "m2"=>59),
                    array("marca"=>"Toyota", "m1"=>110, "m2"=>100)
                );
?>
```

- ¿A qué tipo de array corresponde la siguiente definición?

```php
<?php
    $families = array (
                    "Griffin"=>array ("Peter", "Lois", "Megan"),
                    "Quagmire"=>array("Glenn"),
                    "Brown"=>array("Cleveland", "Loretta", "Junior")
                    );
?>
```

La representación del array anterior sería:

```
Array  (
    [‘Griffin’] => Array (
                        [0] => Peter
                        [1] => Lois
                        [2] => Megan
                    )
    [‘Quagmire’] => Array (
                        [0] => Glenn
                    )
    [‘Brown’] => Array (
                        [0] => Cleveland
                        [1] => Loretta
                        [2] => Junior                        )
)
```

La siguiente sentencia mostraría un único valor del array anterior:

```php
<?php
    echo $families['Griffin'][2];
?>
```

- ¿Cuál es la salida del código anterior?

Ejemplo:

```php
<?php
    $unaFruta['color'] = 'rojo';
    $unaFruta['sabor'] = 'dulce';
    $unaFruta['forma'] = 'redondeada';
    $unaFruta['nombre'] = 'manzana';
?>
```

Ejemplo:

```php
<?php
    $frutas = array(
                "manzana" => array(
                                "color" => "rojo",
                                "sabor" => "dulce",
                                "forma" => "redondeada"),
                "naranja" => array(
                                "color" => "naranja",
                                "sabor" => "ácido",
                                "forma" => "redondeada"),
                "plátano" => array(
                                "color" => "amarillo",
                                "sabor" => "paste-y",
                                "forma" => "aplatanada")
                );
    echo $a["manzana"]["sabor"];
?>
```

- ¿Cuál es la salida del código anterior?

Funciones de interés.

Se propone investigar el funcionamiento de cada una de las funciones presentadas, e incluir otras que puedan ser de interés.

Declaración: **`array(), list()`**. En realidad son constructores del lenguaje y no funciones.

Ordenación: **`asort(), arsort(), ksort(), rsort(), sort(), uasort(), usort(), y uksort().`**

Recorrido: **`each(), next() y prev()`**.

- Completa con otras funciones para trabajar con arrays.

Estandar de codificación. Arrays.

- No se permiten los números negativos como índices.
- Los índices en arrays indexados pueden iniciarse con cualquier valor no negativo, pero se recomienda iniciarlos a cero.
- Para mejorar la legibilidad, se debe agregar un espacio después de cada coma de separación de los índices en la declaración del array.
- Se recomienda hacer la declaración de los arrays asociativos utilizando diferentes líneas.
- Se recomienda dejar espacio antes y después de los operadores de asignación "=>" y de forma que queden alineados.

```php
<?php
    $varArray = array(
                    'priIndice' => 'primerValor',
                    'segIndice' => 'segundoValor'
                );
?>
```

- Se pueden hacer declaraciones de los arrays en varias líneas, siempre y cuando los inicios estén alineados.

```php
<?php
    $meses = array( 1, 2, 3, 4,
                    5, 6, 7, 8,
                    9, 10, 11, 12,
                );
?>
```

ACTIVIDAD GUIADA.

Ejemplos de arrays y funciones anónimas.

- Prueba el funcionamiento del siguiente script.

```php
<?php
    /**
    * Ejemplo de uso de las funciones anónimas.
    *
    */

    $aPaises = array(
       array('id' => 'es', 'pais' => 'España', 'capital' => 'Madrid'),
       array('id' => 'it', 'pais' => 'Italia', 'capital' => 'Roma'),
       array('id' => 'fr', 'pais' => 'Francia', 'capital' => 'Paris'),
    );

    //Obtener un array con los paises.

    //Opción 1
    echo "Opción 1<br/>";
    $nombrePaises = array();
    foreach ($aPaises as $pais) {
        $nombrePaises[] = $pais['pais'];
    }
    print_r ($nombrePaises);

    //Opción 2. Con funciones anónimas.
    echo "<br/>Opción 2<br/>";
    //array_map devuelve un array
    //despues de pasar cada uno de los elementos del array
    //(segundo parámetro)
    //por la función (primer parámetro)
    $nombrePaises = array_map(function ($pais) {
    return $pais['pais'];
    }, $aPaises);

    print_r($nombrePaises);
```

- Crea un script que defina un array de números enteros y utilizando una función anónima genere un array con el cuadrado de los mismos.

ACTIVIDAD GUIADA.

Actualizar la aplicación porfolio utilizando un array para almacenar los datos del porfolio.

- Crea dentro de la estructura de directorios de la aplicación un nuevo directorio de configuración.
- Define y almacena en el directorio de configuración un array con los datos del porfolio.
- Modifica la vista de la aplicación para que utilice los datos almacenados en el array de configuración.

ACTIVIDADES DE CLASE.

1. Definir un array que permita almacenar y mostrar la siguiente información.
    a. Meses del año.
    b. Tablero para jugar al juego de los barcos.
    c. Nota de los alumnos de 2º DAW para el módulo DWES.
    d. Verbos irregulares en inglés.
    e. Información sobre continentes, países, capitales y banderas.
2. Indexar los ejercicios mediante un array.
3. Crear un array con los alumnos de clase y permitir la selección aleatoria de uno de ellos. El resultado debe mostrar nombre y fotografía.
4. Un restaurante dispone de una carta de 3 primeros, 5 segundos y 3 postres. Almacenar información incluyendo foto y mostrar los menús disponibles. Mostrar el precio del menú suponiendo que éste se calcula sumando el precio de cada uno de los platos incluidos y con un descuento del 20 %.
5. Mejorar el calendario con un array de los días festivos: colores diferentes, nacionales, comunidad, locales.

Gestión de formularios.

Referencias:

http://www.php.net

https://developer.mozilla.org/es/docs/Web/HTML/Elemento/form

http://www.w3schools.com

http://www.w3.org/TR/html5/forms.html

Formularios.

Un formulario es el componente necesario en un documento web para que el usuario pueda introducir información.

Sintaxis simplificada:

```
<form
    [ action = ”url_proceso” ]
    [ method = ”GET” |”POST” ]
    [ id  = ”identificador” ]
    [ target = ”destino” ]>
    ...
</form>
```

De los atributos del elemento tienen especial importacia ***action,*** cuyo valor es el nombre del script responsable de procesar el formulario y ***method,*** cuyo valor indica el método utilizado en la petición http.

Algunos de los elementos que se pueden incluir en un formulario son:

input

```
<input type=”tipo” name=”nombre” value=”valor” />
```

Valores para el atributo type: text, radio, password, checkbox, etc.

select

```
<select name=”nombre”>
    <option value=”valor1”> literal </option>
     …
    <option value=”valorn”> literal </option>
</select>
```

textarea

```
<textarea name=”nombre” rows=”filas” cols=”columnas”>

</textarea>
```

Formularios y PHP

Utilizar PHP en el manejo de los formularios nos va a permitir:

- **Procesar** los datos introducidos en el formulario. El valor del atributo action del formulario es el nombre del script responsable del procesamiento.
- Manejar formularios de forma dinámica:
  - Generar todo el formulario.
  - Generar valores iniciales en los campos de entrada.
  - Generar una lista de opciones.

ACTIVIDAD GUIADA.

- Crea un formulario html para introducir el nombre y los apellidos de una persona.
  formulario1.html

```
<form action = "procesa_formulario1.php" method="post">
    <input type="text" name="nombre" placeholder="Nombre" value= ""/>
    <input type="text" name="apellidos" placeholder="Apellidos" value= ""/>
    <input type="submit" name="enviar" value ="Send"/>
</form>
```

- La respuesta al formulario mostrará los datos introducidos.
  procesa_formulario1.php

```
<?php
   echo $_POST['nombre'];
   echo $_POST['apellidos'];
```

- Accede con el navegador a formulario1.html.
- Modifica el formulario para enviar los datos utilizando el método GET.
  formulario2.html

```
<form action = "procesa_formulario2.php" method="get">
    <input type="text" name="nombre" placeholder="Nombre" value= ""/>
    <input type="text" name="apellidos" placeholder="Apellidos" value= ""/>
    <input type="submit" name="enviar" value ="Send"/>
</form>
```

- Crea un script que procese el formulario.
  procesa_formulario2.php

```
<?php
   echo $_GET['nombre'];
   echo $_GET['apellidos'];
```

- Crea un script que almacene la información personal en un array para luego mostrar un formulario.

Procesamiento del formulario.

El nombre del fichero responsable del procesamiento del formulario es el valor del atributo **action** del formulario. En nuestro caso será un archivo PHP. Es posible utilizar el mismo script para mostrar y procesar el formulario. **$_SERVER ["PHP_SELF"]** contiene el nombre del script que se está ejecutando y que podemos utilizar para hacer que el script que procesa el formulario sea el mismo que el que lo presenta. También procesa el formulario el script que lo visualiza cuando al atributo **action** se pone como vacío.

Es conveniente establecer una **capa de control** para determinar bajo que condiciones debe ejecutarse el script. Es evidente, que la respuesta a un formulario sólo tiene sentido cuando es pulsado el botón envío del mismo.

Por defecto todos los campos de un formulario se almacenan automáticamente en un **array asociativo** del script PHP que lo procesa.

La información enviada en los formularios que utilizan el método POST se almacena en el **array asociativo** $**_POST**.

La información enviada en los formularios que utilizan el método GET se almacena en el array asociativo **$_GET**.

El array $**_REQUEST** agrupa el contenido de **$_GET** y **$_POST**.

También es posible recuperar la información introducida en el formulario utilizando la función.

import_request_variables (cadena tipos, cadena prefijo)

```php
<?php
   import_request_variables('P', 'form_');
   echo '$form_nombre = ', $form_nombre, '<br/>'
?>
```

Los índices de los arrays coinciden con el valor del atributo **name** utilizado en el formulario y los **valores** son los datos introducidos por el usuario.

```
<form action = "procesa_formulario1.php" method="post">
    <input type="text" name="nombre" placeholder="Nombre" value= ""/>
```

```php
<?php
   echo $_POST['nombre'];
   echo $_POST['apellidos'];
```

Método GET

La información enviada desde un formulario con el método GET se envía a través de la URL, por tanto, los nombres y valores de las variables son visibles al usuario.

GET también tiene límites en la cantidad de información a enviar. La limitación es de unos 2000 caracteres. Al pasar las variables en la URL es posible marcar la página.

GET puede ser utilizado para enviar datos **no sensibles**. GET NUNCA se debe utilizar para el envío de contraseñas u otra información sensible.

Método POST

La información enviada desde un formulario con el método POST es invisible para los usuarios, los nombres y valores de los campos del formulario se envían en la **petición HTTP**.

No hay límites en la cantidad de información a enviar.

POST soporta funciones avanzadas como la posibilidad de subir archivos al servidor.

Debido a que las variables no se muestran en la URL, no es posible marcar la página.

El contenido de los arrays asociativos que recogen la información del formulario, depende de los tipos de inputs utilizados. De forma general podemos indicar.

- Campos de texto. Las variables asociadas tienen el texto introducido.
- Grupo de botones de opción. La variable asociada contiene el valor del atributo value de la etiqueta input del botón seleccionado.
- Casilla de verificación. La variable asociada contiene el valor contenido en el atributo value de la etiqueta input.
- Lista de selección única. La variable asociada contiene el valor contenido en el atributo value de la etiqueta option, o, si no hay atributo value, el valor mostrado en la lista. **Es posible utilizar el atributo value para crear un código que se almacena en la base de datos, mostramos un nombre, pero almacenamos el código**. En este caso el interfaz de usuario (capa de presentación) debe conocer los códigos utilizados, lo cual no es una buena idea. La solución consiste en generar dinámicamente el formulario a partir de la base de datos.
- Lista de selección múltiple. La variable asociada contiene el valor del atributo value de la etiqueta option. Si no hay atributo value, el valor mostrado en la lista. Esto es válido sólo para la última opción seleccionada, si la variable es una variable escalar. En consecuencia, para obtener una lista de selección múltiple, es necesario utilizar un array.
- Botón de validación. PHP crea una viariable que lleva el nombre del botón y tiene como valor el del atributo value, sólo si se pulsa el botón.
- Botón de imagen. PHP crea dos variables que llevan el nombre del botón (atributo name) seguido de _x y _y y dando la posición relativa, en píxels, del clic con respecto al ángulo situado en la parte superior izquierda de la imagen. Si en botón no tiene ningún nombre, las dos variables se llaman x e y.
- Botón “reset” o “button”. Sólo permiten realizar una acción simple del lado del cliente.

Generación dinámica de formularios.

Para crear formularios de manera dinámica utilizaremos PHP para escribir las cadenas con el contenido HTML necesario. Uno de los aspectos más significativos, es el nombre (valor para el atributo **name**) que debemos utilizar al generar los campos del formulario. En muchos casos será necesario utilizar una notación tipo array para no tener problemas en el proceso de recuperación. PHP completa el array añadiendo una línea para cada campo, con un índice entero consecutivo comenzado en 0. Los índices se asignan en función del orden en el que aparecen los campos en el formulario. Puede aparecer un problema si el orden de los campos cambia. En algunos casos

puede evitarse este problema asignando explícitamente el índice, bien como número o como cadena de caracteres.

```
for ($i=0;$i<$ctdNumeros;$i++){
    echo "<input type=\"number\" name=\"numeros[]\"
```

Control de datos.

El script encargado de procesar la información introducida en un formulario necesita una capa de control que impida que su ejecución si no es lanzado en respuesta al botón de envío del formulario.

```
if (!isset($_POST['enviar']) {
   echo "Acceso no autorizado" // Posible redirección.
}
```

Esa estructura también es posible utilizarla para discriminar dentro de un mismo script la presentación de un formulario o su procesamiento.

```
if (!isset($_POST['enviar']) {
   // presentación del formulario.
}
else {
   // procesamiento del formulario.
}
```

Una vez introducidos los datos es necesario comprobar que son correctos. Uno de los principales objetivos de cualquier desarrollo web será el garantizar la seguridad del sistema. La validación y el manejo adecuado de los datos contribuirá a conseguir dicho objetivo.

El primer control de datos debe realizarse en el lado del cliente, reduciendo así el número de peticiones al servidor, pero no será suficiente. Es necesario establecer mecanismos de control a nivel de cliente, a nivel de servidor y a nivel de bases de datos.

Para realizar una correcta validación de datos a nivel de servidor es necesario que el script que procesa el formulario y valida los datos sea el mismo que el que lo presenta.

El código sería:

```
<form method="post"
      action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>"
>
```

$_SERVER es un array asociativo que contiene información, tales como cabeceras, rutas y ubicaciones de script. Las entradas de este array son creadas por el servidor web. **$_SERVER ["PHP_SELF"]** contiene el nombre del script que se está ejecutando.

htmlspecialchars() es una función que convierte caracteres especiales a entidades HTML. Esto significa que va a reemplazar caracteres HTML, como < y > por < >.

$ _SERVER ["PHP_SELF"] puede ser utilizado por los hackers para realizar ataques Cross Site Scripting (XSS). Cross site scripting (XSS) es un tipo de vulnerabilidad de seguridad informática que se encuentran típicamente en las aplicaciones Web. XSS permite a los atacantes inyectar secuencias de comandos del lado del cliente en las páginas web.

Si PHP_SELF se utiliza en una página, entonces el usuario puede introducir una barra (/) y algo más de Cross Site Scripting (XSS).

Supongamos que tenemos el siguiente formulario en una página llamada “test_form.php”:

```
<form method=”post”
      action=”<?php echo $_SERVER[”PHP_SELF”];?>”
>
```

Si un usuario introduce la URL normal en la barra de direcciones

http://www.example.com/test_form.php,

el código anterior se traduce a:

```
<form method=”post” action=”test_form.php”>
```

Sin embargo, si un usuario introduce como URL la siguiente dirección:

http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E

El código será traducido a:

```
<form method=”post” action=”test_form.php/”><script>alert(‘hacked’)</script>
```

Este código añade una etiqueta de script y un comando de alerta. Y cuando se carga la página, se ejecutará el código JavaScript (el usuario verá un cuadro de alerta). Debemos ser conscientes de que cualquier código JavaScript se puede agregar dentro de la etiqueta <script>. Un hacker puede redirigir al usuario a un archivo en otro servidor, y ese archivo puede contener código malicioso que puede alterar las variables globales o enviar el formulario a otra dirección para guardar los datos introducidos.

Podemos evitar los **$_SERVER [”PHP_SELF”]** exploits mediante el uso de la función **htmlspecialchars**().

El código del formulario debería ser:

```
<form method=”post”
      action=”<?php echo htmlspecialchars($_SERVER[”PHP_SELF”]);?>”
>
```

La función **htmlspecialchars** () convierte caracteres especiales a entidades HTML evitando el XSS.

El código se traduciría en:

```
<form method=”post” action=”test_form.php/&quot;&gt;&lt;script&gt;alert(‘hacked’)&lt;/script&gt;”>
```

Y el intento de exploit falla.

Validación de formularios con PHP

La validación del formulario requerirá:

1. Limpiar los datos de entada.
   - Sustituir caracteres html por sus correspondientes entidades con la función **htmlspecialchars**(). Si un usuario intenta enviar el siguiente contenido en un campo de texto:

     ```
     <script>location.href('http://www.iesgrancapitan.org')</script>
     ```

     No se ejecutará ya que se traducirá en:

     ```
     <script> location.href ('http://www.hacked.com') </script>
     ```

   - Eliminar caracteres innecesarios (espacios, tabulación, nueva línea) con la función **trim**().
   - Eliminar las barras invertidas (\) de los datos de entrada con la función **stripslashes**()

   Lo más conveniente es crear una una función de usuario que se encargue de estas tareas.

2. Validación de los datos.

PHP permite la utilización de expresiones regulares para la validación de los datos de entrada: rangos, fechas, formatos específicos, etc.

Específicamente para la validación, PHP dispone de una serie de filtros definidos por un identificador, un nombre y posibles opciones y los indicadores que definen el comportamiento del filtro. Cada opción está definida por un nombre que se utiliza como clave en un array asociativo. Cada indicador se define por una constante; para especificar varios indicadores, basta con sumar las constantes correspondientes.

Los filtros se pueden utilizar en las funciones **filter_var, filter_var_array, filter_input y filter_input_array**

Los filtros se pueden agrupar en:

Filtros de validación.

Los filtros de validación se utilizan para validar la informaión introducida

```php
<?php
    $correo=$_POST['email'];
    if(!filter_var($correo, FILTER_VALIDATE_EMAIL))
        echo "El correo introducido no es válido";
?>
```

Algunos filtros para validar:

FILTER_VALIDATE_BOOLEAN

FILTER_VALIDATE_EMAIL

FILTER_VALIDATE_FLOAT

FILTER_VALIDATE_INT

FILTER_VALIDATE_IP

FILTER_VALIDATE_REGEXP

FILTER_VALIDATE_URL

Filtros de saneamiento.

Los filtros de saneamiento sirven para corregir información introducida erróneamente.

```php
$url = "http://www.google.com";
// Eliminar cualquier carácter que pueda dar problemas
$url = filter_var($url, FILTER_SANITIZE_URL);
// Luego validamos la URL
if (!filter_var($url, FILTER_VALIDATE_URL) === false) {
    echo("$url es una URL válida");
}
else {
    echo("$url no es una URL válida");
}
```

Algunos filtros para corregir:

FILTER_SANITIZE_EMAIL

FILTER_SANITIZE_ENCODED

FILTER_SANITIZE_MAGIC_QUOTES

FILTER_SANITIZE_NUMBER_FLOAT

FILTER_SANITIZE_NUMBER_INT

FILTER_SANITIZE_SPECIAL_CHARS

FILTER_SANITIZE_STRING

FILTER_SANITIZE_STRIPPED

FILTER_SANITIZE_URL

FILTER_UNSAFE_RAW

Ver documentación php.net

Redireccionamiento

Es posible redirigir al usuario a otra página desde el script utilizando la función **header**()

```php
header('location: url absoluta o relativa');
```

Es necesario utilizarla antes del envío de las cabeceras http. Si la característica de almacenamiento en búffer de la página está activada con la directiva de configuración **output_buffering**, el resultado del script no se envía progresivamente, sino que se coloca en un búffer y se envía de forma completa posteriormente.

Otras funciones relacionadas:

header_list: Lista de encabezados de la respuesta.

header_sent: Permite comprobar si los encabezados ya han sido enviados.

get_headers: Lista de los encabezados reenviados pur un servidor, para una URL determinada.

ACTIVIDAD GUIADA.

1. Crear y procesar un formulario que incluya diferentes tipos de inputs y validación de datos de entrada. El procesamiento consistirá en mostrar los datos introducidos en el formulario.

Actividades de clase.

2. Modifica el ejercicio del calendario para que el mes y el año se lean en un formulario. Añade las siguientes especificaciones:
    a. Por defecto se carga el mes y año actual.
    b. Definición de días festivos en un array.
    c. Utilizar colores para diferenciar festivos nacionales, de comunidad y locales.
    d. Cada día será un enlace a una página que mostrará la fecha seleccionda.
3. Formulario para crear un currículum que incluya: Campos de texto, grupo de botones de opción, casilla de verificación, lista de selección única, lista de selección múltiple, botón de validación, botón de imagen, botón de reset, etc.
4. Crear una aplicación que almacene información de países: nombre capital y bandera. Diseñar un formulario que permita seleccionar un país y nos muestre el nombre de la capital y la bandera.
5. Crear un script para sumar una serie de números. El número de números a sumar será introducido en un formulario.