PHP: variables, tipos, operadores, expresiones, estructuras de control

Ejercicio 1

```php
<?php
function doble($i) {
   return $i*2;
}
$a = TRUE;
$b = "xyz";
$c = 'xyz';
$d = 12;
echo gettype($a);
echo gettype($b);
echo gettype($c);
echo gettype($d);
if (is_int($d)) {
   $d += 4;
}
if (is_string($a)) {
   echo "Cadena: $a";
}
$d = $a ? ++$d : $d*3;
$f = doble($d++);
$g = $f += 10;
echo $a, $b, $c, $d, $f , $g;
?>
```

- **Variables y su tipo:**
  a: boolean
  b: string
  c: string
  d: integer
  f: integer
  g: integer

- **Operadores:**
  - **=** (Operador asignación)
  - ***** (Operador aritmético)
  - **+=** (Operador combinado)
  - **++** (Operador de incremento. Se utiliza tanto de pre-incremento como de post-incremento)

- **Funciones y sus parámetros**
    - **gettype (mixed $var) : string**
      Retorna el tipo de variable PHP de var.
    - **is_int (mixed $var) : bool**
      Comprueba si el tipo de variable dado es de tipo integer.
    - **is_string (mixed $var) : bool**
      Comprueba si el tipo de variable dado es de tipo string.
    - **doble($var):**
      Retorna el doble de lo ingresado en el parámetro.

      La palabra clave *mixed* indica que el parámetro acepta múltiples tipos.

- **Estructuras de control**
    - **if (expresión) :** La expresión es evaluada a su valor booleano. Si la expresión se evalúa como TRUE, PHP ejecutará la sentencia y si se evalúa como FALSE la ignorará.
    - **return argumento :** Al ser llamada desde una función, termina la ejecución de la misma y devuelve su argumento como el valor de la llamada a la función.

      ***return** es un **constructor del lenguaje** y no una **función**, los paréntesis que rodean su argumento no son obligatorios y se desaconseja su uso.*

- **Salida por pantalla**
      booleanstringstringinteger1xyzxyz184444

Ejercicio 2

A. Los códigos son equivalentes, los tres generan la misma salida en pantalla.
12345678910

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

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

```php
<?php
 $i = 0;
 do {
    print ++$i;
 } while ($i<10);
?>
```

En los dos primeros cuadros se utilizan secuencias alternativas del ***while***.
En el primero de ellos se muestra en pantalla la variable **$i** y después se le hace un incremento de 1.
En el segundo se hace lo mismo, solo que el incremento se hace en otra línea.

El tercer cuadro utiliza una estructura de control ***do-while***.
En este ejemplo se utiliza el incremento antes de mostrar la variable en pantalla.

B. Los códigos son equivalentes, todos generan la misma salida en pantalla.
12345678910

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

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

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

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

En los códigos de la primer columna se hace lo mismo, solo que en el **primero**, la sentencia que se ejecuta esta en el **bloque delimitado por las llaves**, mientras que en el **segundo** está **incluido en el tercer parámetro** (El cual se ejecuta luego de cada iteración)

En el código de la **segunda columna**, el bucle se corre **indefinidamente**, por lo que se utiliza dentro un **break** para cortar dicha ejecución cuando se alcance una condición.

En la **tercer columna**, el código recibe **parámetros vacíos**, por lo cual el bucle corre **indefinidamente**, hasta que al llegar a un número mayor que 10, corta su ejecución mediante un **break**.

C. Ambos códigos son equivalentes, teniendo la misma salida por pantalla.

```
<?php
…
…
if ($i == 0) {
   print "i equals 0";
} elseif ($i == 1) {
   print "i equals 1";
} elseif ($i == 2) {
   print "i equals 2";
}
?>
```

```
<?php
…
…
switch ($i) {
   case 0:
      print "i equals 0";
      break;
   case 1:
      print "i equals 1";
      break;
   case 2:
      print "i equals 2";
      break;
}
?>
```

En el primero se utiliza la estructura de ***if / elseif*** para mostrar en pantalla el resultado deseado.

En el segundo se utiliza la estructura ***switch*** para evaluar la variable **$i** y dependiendo el caso, mostrar en pantalla el resultado deseado. Luego se vale de un **break** sino el **case** seguiría hacia abajo, mostrando en pantalla otros resultados.

Ejercicio 3

A.

```
<html>
<head><title>Documento 1</title></head>
<body>
<?php
   echo "<table width = 90% border = '1' >";
   $row = 5;
   $col = 2;
   for ($r = 1; $r <= $row; $r++) {
         echo "<tr>";
         for ($c = 1; $c <= $col;$c++) {
               echo "<td> </td>\n";
         }     echo "</tr>\n";
   }
   echo "</table>\n";
?>
</body></html>
```

El código crea una tabla de 5 filas y 2 columnas, de manera dinámica mediante dos iteraciones anidadas con for. El mismo tiene esta salida por pantalla.

B.

```
<html>
<head><title>Documento 2</title></head>
<body>
<?php
if (!isset($_POST['submit'])) {
?>
   <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
   Edad: <input name="age" size="2">
   <input type="submit" name="submit" value="Ir">
   </form>
<?php
   }
else {
   $age = $_POST['age'];
   if ($age >= 21) {
      echo 'Mayor de edad';
      }
   else {
      echo 'Menor de edad';
   }
}
?>
</body></html>
```

El siguiente código muestra el formulario si aún no fue enviado, lo cual determina mediante la utilización de la función ***isset***.

El *action* estará dirigido a la ruta del script que se está ejecutando.

En caso de que ya haya sido enviado, muestra si es mayor o menor de edad, en base a la edad ingresada en el formulario.

Mayor de edad

Ejercicio 4

datos.php

```
<?php
$color = 'blanco';
$flor = 'clavel';
?>
```

```
<?php
echo "El $flor $color \n";
include 'datos.php';
echo " El $flor $color";
?>
```

Salida:

El El clavel blanco

En primer lugar las variables *$flor* y *$color* se encuentran vacías, dado que al ser secuencial, aún no estaban importadas con el **include 'datos.php'.**

Luego de ser incluido el archivo que inicializa esas variables, termina mostrándose en pantalla **El clavel blanco.**

El salto de linea no es tomado en cuenta hasta que no se utiliza este header:
header('Content-type: text/plain');

```
El
El clavel blanco
```

Ejercicio 5

contador.php

```
<?
// Archivo para acumular el numero de visitas
$archivo = "contador.dat";
// Abrir el archivo para lectura
$abrir = fopen($archivo, "r");
// Leer el contenido del archivo
$cont = fread($abrir, filesize($archivo));
// Cerrar el archivo
fclose($abrir);
// Abrir nuevamente el archivo para escritura
$abrir = fopen($archivo, "w");
// Agregar 1 visita
$cont = $cont + 1;
// Guardar la modificación
$guardar = fwrite($abrir, $cont);
// Cerrar el archivo
fclose($abrir);
// Mostrar el total de visitas
echo "<font face='arial' size='3'>Cantidad de visitas:".$cont."</font>";
?>
```

visitas.php

```
<!-- Página que va a contener al contador de visitas  -->
<html>
<head></head>
<body>
<? include("contador.php")?>
</body>
</html>
```

En ambos archivos, luego de los **<?** debi incluir la palabra **php** .

En primera instancia se crea un archivo **contador.dat** con el valor inicial de las visitas.
Cuando se ingresa a **visitas.php** se ejecuta el código de **contador.php** el cual abre el dat creado en modo lectura y lee el valor existente.
Después de eso, vuelve a abrirlo en modo escritura y actualiza la cantidad, retornando al final el elemento html que nos muestra la cantidad de visitas.

La cantidad de visitas se actualizan cada vez que se recarga la página.

Cantidad de visitas:1

Cantidad de visitas:2

Al ser un archivo, si más de una persona ingresa al mismo tiempo a la página, puede presentarse una inconsistencia de datos ya que no permiten concurrencia.

PHP: arrays, funciones

Ejercicio 1:

```php
<?php
$a = array( 'color'  => 'rojo',
          'sabor'  => 'dulce',
          'forma'  => 'redonda',
          'nombre' => 'manzana',
          4
        );
?>
```

```php
<?php
$a['color']  = 'rojo';
$a['sabor']  = 'dulce';
$a['forma']  = 'redonda';
$a['nombre'] = 'manzana';
$a[]         = 4;
?>
```

Los códigos son equivalentes. Ambos son arrays asociativos, la diferencia está en que desde el primero se crea el array desde el constructor del lenguaje **array()** y en el segundo se utiliza otra sintaxis.

Ejercicio 2:

a)

```php
<?php
$matriz = array("x" => "bar", 12 => true);
echo $matriz["x"];
echo $matriz[12];
?>
```

b)

```php
<?php
$matriz = array("unamatriz" => array(6 => 5, 13 => 9, "a" => 42));

echo $matriz["unamatriz"][6];
echo $matriz["unamatriz"][13];
echo $matriz["unamatriz"]["a"];
?>
```

c)

```php
<?php
$matriz = array(5 => 1, 12 => 2);
$matriz[] = 56;
$matriz["x"] = 42;  unset($matriz[5]);  unset($matriz);
?>
```

salida a) **bar1**
salida b) **5942**
salida c) *No tiene retorno.*

Ejercicio 3:

a)

```
<?php
$fun = getdate();
echo "Has entrado en esta pagina a las $fun[hours] horas, con $fun[minutes] minutos y $fun[seconds]
segundos, del $fun[mday]/$fun[mon]/$fun[year]";
?>
```

b)

```
<?php
function sumar($sumando1,$sumando2){
  $suma=$sumando1+$sumando2;
  echo $sumando1."+".$sumando2."=".$suma;
}
sumar(5,6);
?>
```

a) **Has entrado en esta pagina a las 21 horas, con 22 minutos y 59 segundos, del 20/5/2020**
b) **5+6=11**

Ejercicio 4:

```
function comprobar_nombre_usuario($nombre_usuario){
  //compruebo que el tamaño del string sea válido.
  if (strlen($nombre_usuario)<3 || strlen($nombre_usuario)>20){
    echo $nombre_usuario . " no es válido<br>";
    return false;
  }

  //compruebo que los caracteres sean los permitidos
  $permitidos = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-
_";
  for ($i=0; $i<strlen($nombre_usuario); $i++){
    if (strpos($permitidos, substr($nombre_usuario,$i,1))===false){
      echo $nombre_usuario . " no es válido<br>";
      return false;
    }
  }
  echo $nombre_usuario . " es válido<br>";
  return true;
}
```

Al código anterior se le agregaron las etiquetas de PHP y se procedió a probar la función.

Dicha función valida un nombre de usuario según una serie de reglas:

1. Valida la **cantidad mínima de caracteres** (Debe ser mayor que 2).
2. Valida la **cantidad máxima de caracteres** (Debe ser menor que 21).
3. Valida los caracteres permitidos.

Primero corrobora la longitud del nombre usuario a comprobar y por último que no posea ningún caracter que esté por fuera de los permitidos.

Ejemplos:

Comprobamos con **caracteres alfabéticos** permitidos..
Entrada >>> comprobar_nombre_usuario(‘pladreyt’)
Salida >>> pladreyt es válido

Comprobamos con **caracteres alfanuméricos** permitidos.
Entrada >>> comprobar_nombre_usuario(‘pladreyt1993’)
Salida >>> pladreyt1993 es válido
Entrada >>> comprobar_nombre_usuario(‘pladreyt-1993’)
Salida >>> pladreyt-1993 es válido

Comprobamos con **caracteres numéricos** permitidos.
Entrada >>> comprobar_nombre_usuario(‘1993’)
Salida >>> 1993 es válido

Comprobamos cuando excede la **longitud máxima**.
Entrada >>> comprobar_nombre_usuario(‘pabloladreyt1993rosario’)
Salida >>> pabloladreyt1993rosario no es válido

Comprobamos cuando no alcanza la **longitud mínima.**
Entrada >>> comprobar_nombre_usuario(‘pa’)
Salida >>> pa no es válido

Comprobamos con **caracteres** no permitidos.
Entrada >>> comprobar_nombre_usuario(‘pabloladreyt**2’)
Salida >>> pabloladreyt**2 no es válido