50 ANIVERSARIO

IES ANTONIO SEQUEROS ALMORADÍ

TEMA 3

Estructuras de Control

2020/2021

Versión: 201007.2251

IES Antonio Sequeros

ÍNDICE

1. OPERACIONES LÓGICAS

Las operaciones lógicas son expresiones matemáticas cuyo resultado es un valor booleano (verdadero o falso ≡ true o false).

Estas expresiones se utilizan principalmente en las estructuras de control.

1.1. Comparaciones

Las comparaciones permiten comparar variables o expresiones entre sí o con valores concretos. El resultado de la comparación es un valor booleano (true o false).

Ejemplo	*Nombre*	*Resultado*
$a == $b	Igual	true si $a es igual a $b.
$a === $b	Idéntico	true si $a es igual a $b, y son del mismo tipo. (a partir de PHP 4)
$a != $b	Diferente	true si $a no es igual a $b.
$a <> $b		
$a !== $b	No idénticos	true si $a no es igual a $b, o si no son del mismo tipo. (a partir de PHP 4)
$a < $b	Menor que	true si $a es estrictamente menor que $b.
$a > $b	Mayor que	true si $a es estrictamente mayor que $b.
$a <= $b	Menor o igual que	true si $a es menor o igual que $b.
$a >= $b	Mayor o igual que	true si $a es mayor o igual que $b.

1.2. Operadores lógicos

Los operadores lógicos permiten combinar expresiones simples en expresiones más complejas.

Ejemplo	*Nombre*	*Resultado*
$a && $b	Y	true si los dos, $a y $b, son true.
$a and $b		
$a || $b	O	true si uno de los dos, $a o $b, es true.
$a or $b		
$a xor $b	O exclusivo (Xor)	true si sólo uno de los dos, $a o $b, es true, pero no ambos.
! $a	Negación	true si $a no es true.

Al escribir expresiones en las que se combinan varias comparaciones mediante operadores lógicos es conveniente utilizar paréntesis, aunque en muchos casos no sean necesarios porque las comparaciones tienen precedencia sobre los operadores lógicos.

1.2.1 Diferencia entre and y && y entre or y ||

Los operadores and y && y los operadores or y || no son completamente equivalentes, ya que no tienen la misma precedencia. Concretamente, && y || tienen mayor prioridad que and y or. Como además el operador de asignación = tiene una prioridad intermedia, se pueden producir situaciones inesperadas, como muestran los siguientes ejemplos.

<?php
$var1 = true;
$var2 = false;
$todo = $var1 && $var2;
if ($todo) {
print " verdadero \n";
} else {
print " falso \n";
}
?>
	falso

<?php
$var1 = true;
$var2 = false;
$todo = $var1 and $var2;
if ($todo) {
print " verdadero \n";
} else {
print " falso \n";
}
?>
	verdadero

2. ESTRUCTURAS DE CONTROL

2.1. BIFURCACIÓN

2.1.1 if ... elseif ... else

Las construcciones if, else y elseif permiten condicionar la ejecución de un bloque de sentencias al cumplimiento de una condición.

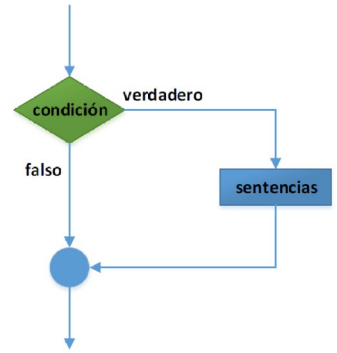
La condición se evalúa **siempre**, si el resultado es **true** se ejecuta el bloque de sentencias y si el resultado es **false** no se ejecuta el bloque de sentencias.

La sintaxis de la construcción if sencilla es la siguiente:

if

```
if (condición) {
    bloque_de_sentencias
}
```

```
if (condicion)
{
   sentencias
}
```



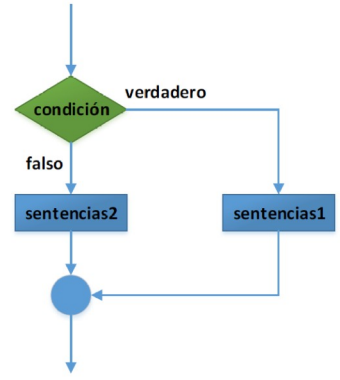
A la construcción if se le puede añadir la instrucción else:

La condición se evalúa siempre, si el resultado es true se ejecuta solamente el bloque de sentencias 1 y si el resultado es false se ejecuta solamente el bloque de sentencias 2.

If else

```
if (condición) {
    bloque_de_sentencias_1
} else {
    bloque_de_sentencias_2
}
```

```
if (condicion)
{
   sentencias1
}
else
{
   sentencias2
}
```



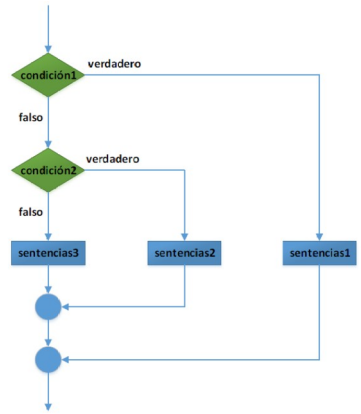
La construcción if ... else se puede extender añadiendo la instrucción elseif:

La condición 1 se evalúa siempre, si el resultado es true se ejecuta solamente el bloque de sentencias 1 y solamente si el resultado es false se evalúa la condición 2, si el resultado de esta es true se ejecuta solamente el bloque de sentencias 2 y si el resultado es false se ejecuta solamente el bloque de sentencias 3. En cualquier caso solamente se ejecuta uno de los tres bloques de sentencias.

If elseif else

```
if (condición_1) {
    bloque_de_sentencias_1
} elseif (condición_2) {
    bloque_de_sentencias_2
} else {
    bloque_de_sentencias_3
}
```

```
if (condicion1)
{
   sentencias1
}
else
{
   if (condicion2)
   {
      sentencias2
   }
   else
   {
      sentencias3
   }
}
```

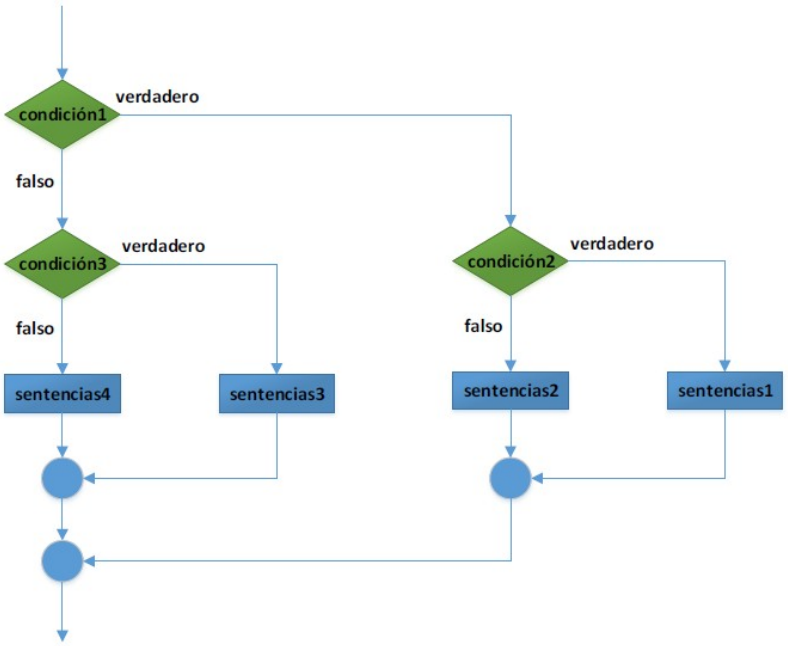


Se pueden añadir tantas instrucciones elseif como se desee, teniendo en cuenta que en cualquier caso solamente se ejecuta uno de los bloques de sentencias.

También pueden anidarse:

If else múltiple

```
if (condicion1)
{
   if (condicion2)
   {
      sentencias1
   }
   else
   {
      sentencias2
   }
}
else
{
   if (condicion3)
   {
      sentencias3
   }
   else
   {
      sentencias4
   }
}
```



Notación abreviada .. ? ... : ...

If else	.. ? ... : ...
`if (expresión_1) {`	`$variable = (expresión_1) ? expresion1`

```
    $variable = expresion1;
} else
    $variable = expresion2;
}
```

```
: expresion2;
```

Las dos construcciones son equivalentes

2.1.2 switch

La sentencia switch es equivalente a una construcción if ... elseif ... en las que las expresiones son comparaciones de igualdad de la misma expresión con valores distintos.

La sintaxis de la sentencia switch es la siguiente:

switch

```
switch (expresión_1) {
    case valor_1:
        bloque_de_sentencias_1;
        break;
    case valor_2:
        bloque_de_sentencias_2;
        break;
    ...
    case valor_n:
        bloque_de_sentencias_n;
        break;
    default:
        bloque_de_sentencias_default;
}
```

```
if (expresión_1 == valor_1) {
    bloque_de_sentencias_1
} elseif (expresión_1 == valor_2) {
    bloque_de_sentencias_2
} elseif (expresión_1 == valor_3) {
    bloque_de_sentencias_3
 ...
} elseif (expresión_1 == valor_n) {
    bloque_de_sentencias_n
}
  else {
    bloque_de_sentencias_default;
}
```

Es importante entender cómo la sentencia *switch* es ejecutada con el fin de evitar errores. La sentencia *switch* ejecuta línea por línea (en realidad, sentencia por sentencia).

- Al principio, ningún código es ejecutado.
- Solo cuando se encuentra una sentencia *case* cuya expresión se evalúa a un valor que coincida con el valor de la expresión *switch*, PHP comienza a ejecutar la sentencias.
- PHP continúa ejecutando las sentencias hasta el final del bloque *switch*, o hasta la primera vez que vea una sentencia *break*. Si no se escribe una sentencia *break* al final de la lista de sentencias de un caso, PHP seguirá ejecutando las sentencias del caso siguiente.

Por ejemplo:

```
<?php
switch ($i) {
    case 0:
        echo "i es igual a 0";
    case 1:
        echo "i es igual a 1";
    case 2:
        echo "i es igual a 2";
}
?>
```

```
Aquí, si $i es igual a 0, PHP
ejecutaría todas las sentencias echo!
Si $i es igual a 1, PHP ejecutaría las
últimas dos sentencias echo.
Se obtendría el comportamiento
esperado (se mostraría 'i es igual a
2') sólo si $i es igual a 2.
Por lo tanto, es importante no olvidar
las sentencias break (aunque es
posible que se desee evitar
proporcionarlas a propósito bajo
determinadas circunstancias).
```

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

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

Práctica 03

1. Parimpar.php → Dado un número entero positivo determine si es par o impar.

2. Mayor2num.php → Dados dos números ($a y $b) mostrar un mensaje diciendo cual de ellos es mayor

3. Compara2num.php → Dados dos números ($a y $b) mostrar un mensaje diciendo si a es mayor, menor o igual a b

4. TresCuadrados.php → Escriba un programa que cada vez que se ejecute calcule tres cuadrados y nos muestre el valor de su lado, su área y nos diga qué cuadrado es el mayor.

5. Diasemana.php → Dado un número del 1 al 7 indique a que día de la semana corresponde

6. Seguros.php → Dada la siguiente tabla de precios de una compañía aseguradora:

```
100 € - Jóvenes Sanos
120 € - Jóvenes Enfermos – Edad Media Sanos
140 € - Edad Media Enfermos – Edad Avanzada Sanos
No se admiten – Edad Avanzada Enfermos

Donde se considera:
Gente Joven a menores de 28 años
Gente de Edad Media entre 28 y 50 años
Gente de Edad Avanzada a mayores de 50 años
```

   Escribir un programa que pida edad y estado de Salud (S/E) y nos devuelva el importe de la póliza.

7. Radar.php → En la nueva normativa de tráfico una velocidad superior al 50% a la permitida, siempre que el exceso sea mayor a 30 km /h, supone la retirada del carnet de conducir. Escribir un programa que dada la velocidad máxima permitida y la velocidad de circulación muestre si se debe retirar el carnet.

8. Billetes.php → Escribe un programa que determine el menor número de billetes y monedas de curso legal (euros) equivalente a cierta cantidad de dinero (1, 2, 5, 10, 20, 50, 100, 200, 500 euros)

9. Bisiesto.php → Calcular si un año es bisiesto. La regla completa para saber si un año es bisiesto, es que sea divisible por 4, excepto que si es divisible por 100 lo sea también por 400. Realizar la comprobación mediante una sola operación lógica e imprima en la pantalla un mensaje con el resultado.

   Para saber si un año es bisiesto podemos utiliza una de las siguientes:

   - EsABisiesto = ($ano % 4) == 0 and (($ano % 100) != 0 or ($ano % 400) == 0)
   - EsABisiesto = ($ano%4==0 && $ano%100!=0) || $ano%400==0

10. Fechacorrecta.php → Dado tres número decir si forman una fecha exacta (no tenemos en cuenta los años bisiestos). Al no tener en cuenta si el año es bisiesto o no, solo tendremos en cuenta el día y el mes para comprobar si la fecha es correcta.

11. FechacorrectaBis.php → Dado tres número decir si forman una fecha exacta (**SI** tenemos en cuenta los años bisiestos)

12. Ordenar3num.php → Dados 3 números aleatorios mostradlos en orden ascendente

13. triangulo.php → Las longitudes de los lados de un triángulo han de cumplir que la suma de dos de ellas ha de ser mayor que la otra (de otra manera, cada uno de ellos debe ser menor que la suma de los otros dos). Cuando los tres lados son distintos entre sí el triángulo es escaleno, cuando los tres lados son iguales el triángulo es equilátero y si al menos dos lados son iguales el triángulo es isósceles. Haz un programa que solicite tres cantidades reales positivas e indique si dichos valores pueden formar un triángulo y el tipo del triángulo.

14. Funciontrozos.php → Escribe un programa que dado un valor x calcule su valor asociado f(x) según la función:

$$f(x)=\begin{cases} x^3-x^2+6 & si\ x>0 \\ -5 & si\ x=0 \\ \dfrac{}{\sqrt{|x^3-x|}} & si\ x<0 \end{cases}$$

2.2. Iteración o Bucle

Todos los lenguajes de programación poseen instrucciones que permiten que una instrucción o bloque de instrucciones se ejecuten un número determinado de veces, tantas como deseemos.

Veamos un ejemplo:

Sumar un número, n, de veces un valor, num, (o lo que es lo mismo num * n), pero supongamos que no podemos multiplicar, sino que hemos de resolverlo solo con sumas.

num + num + num + num + num + num+ … + num (n sumandos)

Lo puedo resolver asi:

```
$suma = 0;
$suma = $suma + $num      // 1ª suma, ahora $suma vale $num
$suma = $suma + $num      // 2ª suma, ahora $suma vale $num+$num
$suma = $suma + $num      // 3ª suma, ahora $suma vale $num+$num+$num
…
$suma = $suma + $num      // nª suma, ahora $suma vale $num+$num+ … +$num (n veces)
```

Hemos repetido n veces la misma instrucción, por lo que si usamos un bucle que se repita n veces lo podemos hacer así:

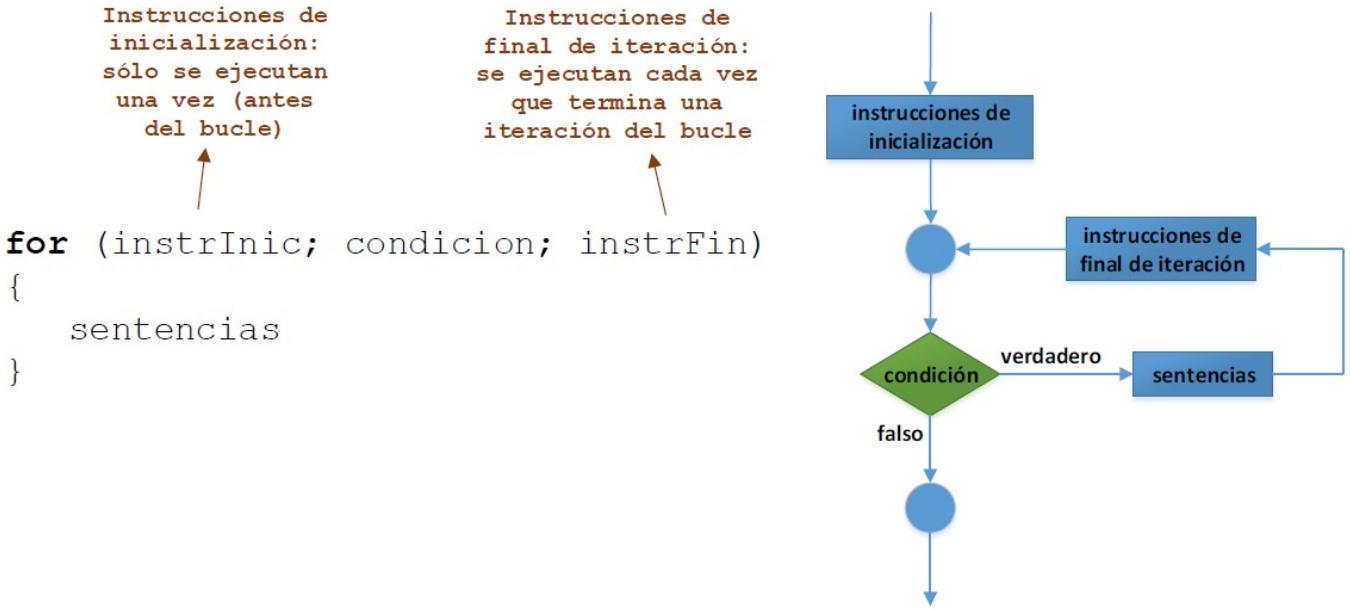
```
$suma = 0;
repetir n veces
  $suma = $suma + $num
finrepeticion
```

Que solo necesitamos que n tome el valor que queramos no debemos de insertar más código para obtener la suma de num n veces.

2.2.1 for

Sintaxis

```
for (asignación_inicial; condición_continuación; cambio_variable) {
    bloque_de_sentencias
}
```



La ejecución de un bucle **for** se realiza en el siguiente orden:

- Se establece el valor inicial de la variable de control definida en la asignación inicial.
- Se evalúa la condición de continuación:
  - si el resultado es true se ejecuta el bloque de sentencias, se efectúa el cambio de la variable de control y se evalúa nuevamente la condición de continuación;
  - si el resultado es false el bucle se termina.

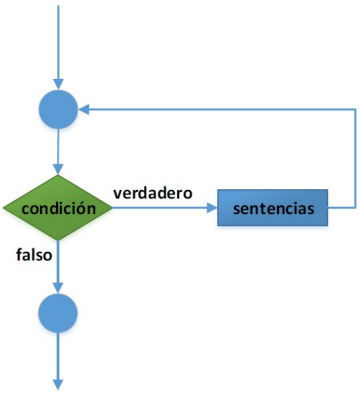
Ejemplo:

```
<?php
for ($i = 0; $i < 5; $i++) {
    print "$i\n";
}
?>
```

```
0
1
2
3
4
```

2.2.2 while

Sintaxis	

`while (expresión) {`
 	
`bloque_de_sentencia`
`s`
`}`
	`while (condicion)`
	`{`
	`  sentencias`
	`}`
	 

La expresión se evalúa al principio de cada iteración: si el resultado es true se ejecuta el bloque de sentencias; si el resultado es false el bucle se termina.

Cuando se programa un bucle while hay que tener cuidado en que la expresión deje de cumplirse en algún momento, porque si no es así, el bucle no terminaría nunca.

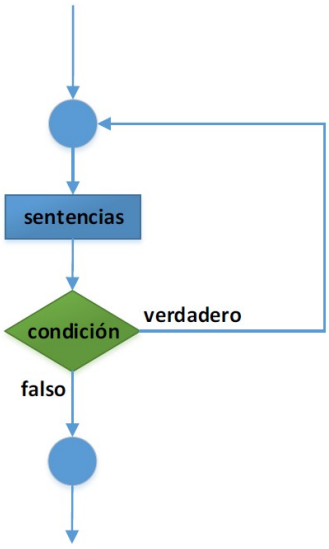
Ejemplo:

```
<?php
$i = 0;
while ($i < 5) {
    print "$i\n";
    $i++;
}
?>
```

```
0
1
2
3
4
```

2.2.3 do ... while

Sintaxis	

`do {`
`  bloque_de_sentencias`
`} while (expresión)`
	`do`
	`{`
	`  sentencias`
	`}`
	`while (condicion);`
	 

La expresión se evalúa al final de cada iteración: si el resultado es true se ejecuta el bloque de sentencias; si el resultado es false el bucle se termina.

El bucle do ... while es muy similar al bucle while, la principal diferencia es que en el bucle do ... while el bloque de sentencias se ejecuta por lo menos una vez mientras que en el bucle while depende de si la expresión es cierta o no la primera vez que se evalúa.

Cuando se programa un bucle do ... while hay que tener cuidado en que la expresión deje de cumplirse en algún momento, porque si no es así, el bucle no terminaría nunca.

Ejemplo:

```
<?php
$i = 0;
do {
    print "<p>$i</p>\n";
    $i++;
} while ($i < 5)
?>
```

```
0
1
2
3
4
```

2.2.4 Bucles Anidados

Bucle anidado: Bucle incluido dentro del bloque de instrucciones de otro bucle principal.

Distinguimos:

- Bucle interior: El que se encuentra dentro del otro.
- Bucle exterior: Al bucle principal.

Existen dos tipos de bucles anidados. Los de variables independientes y los de variables dependientes.

Bucle anidado con variables independientes

Las dos variables son independientes cuando los valores que toma la variable de control del bucle interior no dependen del valor de la variable de control del bucle externo.

Por ejemplo:

```
<?php
for ($i=0;$i<3;$i++)
    for ($j=0;$j<2;$j++)
        echo 'i vale ', $i, ' y j vale ', $j, "<br>" ;
?>
```

```
i vale 0 y j vale 0
i vale 0 y j vale 1
i vale 1 y j vale 0
i vale 1 y j vale 1
i vale 2 y j vale 0
i vale 2 y j vale 1
```

```
<?php
$i=0;
while ($i < 3) {
    $j=0;
    while $j<2) {
        echo 'i vale ', $i, ' y j vale ', $j, "<br>" ;
        $j++;
    }
    $i++;
}
?>
```

```
i vale 0 y j vale 0
i vale 0 y j vale 1
i vale 1 y j vale 0
i vale 1 y j vale 1
i vale 2 y j vale 0
i vale 2 y j vale 1
```

En ambos casos el resultado sería el mismo.

Habitualmente se utiliza la letra i como nombre de la variable de control del bucle exterior y la j como nombre de la variable de control del bucle interior (k si hay un tercer nivel de anidamiento.

Bucle anidado con variables dependientes

Los valores que toma la variable de control del bucle interior dependen del valor de la variable de control del bucle exterior.

Por ejemplo:

```
<?php
for ($i=1;$i<=3;$i++)
    for ($j=0;$j<$i;$j++)
        echo 'i vale ', $i, ' y j vale ',
$j, "<br>" ;
?>
```

```
i vale 1 y j vale 0
i vale 2 y j vale 0
i vale 2 y j vale 1
i vale 3 y j vale 0
i vale 3 y j vale 1
i vale 3 y j vale 2
```

Ordenes break, continue en bucles

Estas órdenes permiten modificar la forma en la que se ejecuta la secuencia de comandos en un bucle for y while. Su función es:

Orden	función
break	Termina la ejecución del bucle más inmediato que la contiene.
continue	Termina la ejecución de la iteración actual pasando a la siguiente.

Por ejemplo:

	<?php
	$i = 1;
	while ($i <= 10) {
	echo "$i - ";
	$i++;
	}
	echo "Programa terminado"
	?>
		1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - Programa terminado

break	<?php
	$i = 1;
	while ($i <= 10) {
	echo "$i - ";
	$i++;
	if ($i==7)
	break;
	}
	echo "Programa terminado"
	?>
		1 - 2 - 3 - 4 - 5 - 6 - Programa terminado

continue	<?php
	$i = 1;
	while ($i <= 10) {
	echo "$i - ";
	$i++;
	if ($i==7)
	continue;
	}
	echo "Programa terminado"
	?>
		1 - 2 - 3 - 4 - 5 - 6 - 8 - 9 - 10 - Programa terminado

Práctica 04

15. Escribir un programa que muestre los N primeros números naturales

    15.a) Utilizando for

    15.b) Utilizando while

16. Escribir un programa que muestre los N primeros números naturales pares

    16.a) Utilizando for

    16.b) Utilizando while

17. Escribir un programa que calcule la suma de los N primeros números naturales

    17.a) Utilizando for

    17.b) Utilizando while

18. Escribir un programa que calcule la suma de N términos de las siguientes series:

    18.a) $1 + 1/2 + 1/3 + \ldots + 1/N$

    18.b) $1 + 4 + 9 + 16 + \ldots + N^2$

    18.c) $1/2 + 1/4 + 1/8 + 1/16 + 1/32 + \ldots + 1/2^N$

    18.d) $1/2 + 2/2^2 + 3/2^3 + ... + N/2^N$.

19. Dado un número n, mostrar por pantalla n filas con los números de 1 a n:

```
1 2 3 ........ n-1 n
1 2 3 ........ n-1 n
.........
.........
1 2 3 ........ n-1 n
1 2 3 ........ n-1 n
```

20. Mostrar en pantalla una tabla de 10 por 10 con los números del 1 al 100

21. Igual al ejercicio anterior, pero colorear las filas alternando gris y blanco. Además, el tamaño será una constante: define(TAM, 10)

22. Escribe el código necesario para obtener la siguiente salida:

```
0123456
1234567
2345678
```

23. Escribir un programa que dado un valor no negativo n, visualice la siguiente salida:

```
1 2 3 ........ n-1 n
1 2 3 ........ n-1
.........
1 2 3
1 2
1
```

24. Muestra los números del 1 al 10 . Sintaxis del bucle for: for(expr1;expr2;expr3)

    24.a) Definiendo las tres expresiones del bucle for

    24.b) Utilizando dos expresiones y la sentencia break

    24.c) Sin expresiones (vacías) y la sentencia break, claro

    24.d) ¿Podrías realizar el ejercicio sin ninguna sentencia fuera de la declaración del bucle?

25. Dada la siguiente serie matemática:

$$a_0 = 0$$
$$a_1 = 1$$
$$a_i = a_{i-1} + 2 * a_{i-2} \quad para \quad i \geq 2$$

    Determinar cuál es el valor y el primer término cuyo valor sea mayor o igual a 2000 (i tal que a_i >= 2000).

26. Utilizando la serie de Taylor para e^x , hacer un programa que calcule dicha función con una precisión dada por el número de términos que se suman (dado un número n, hay n+1 sumandos, al aumentar el valor de n aumentamos a su vez la precisión)

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots\cdots + \frac{x^n}{n!} + \cdots \quad , \forall x \in R$$

27. Escribir un script que dado un valor de n (igual a 5 en la figura) muestre por pantalla la siguiente figura (para n=5):

```
*
**
***
****
*****
```

28. Escribir un script que dado un valor n muestre por pantalla la siguiente figura (para n=5):

```
*
***
*****
*******
*********
```

29. Realizar un programa que dibuje la siguiente figura introduciendo el ancho:

```
Para Ancho=5
*
* *
* * *
* * * *
* * * * *
* * * *
* * *
* *
*
```

30. Realizar un programa que dibuje la siguiente figura introduciendo el ancho y el alto:

```
Alto: 4
Ancho: 6
* * * * * *
*         *
*         *
* * * * * *
```

```
Alto: 5
Ancho: 4
* * * *
*     *
*     *
*     *
* * * *
```

31. Realizar un programa que dibuje la siguiente figura introduciendo el ancho:

```
Ancho: 7
      *
    * * *
  * * * * *
* * * * * * *
```

32. Realizar un programa que dibuje la siguiente figura introduciendo el alto:

```
Alto: 5
* * * * * * * * *
* * * *   * * * *
* * *       * * *
* *           * *
*               *
```

33. Escribir un programa que dado un número nos de el factorial de ese número. El factorial de n se define como el producto de todos los números enteros positivos desde 1 hasta n:

```
n!  = 1 x 2 x 3 x 4 x … x (n-1) x n
10! = 1 x 2 x 3 x 4 x … x 9 x 10
```

34. Escribir un programa que calcule la suma de los dígitos de un número entero. Ejemplo: para 3913 debe devolver 16 (3+9+1+3)

35. Realizar un programa que dado un número de filas muestre el siguiente triángulo, por ejemplo para filas = 11. (observar los números mayores de 9):

```
           1
          232
         34543
        4567654
       567898765
      67890109876
     7890123540987
    890123454321098
   90123456765432109
  0123456789876543210
1234567890109876543210
```

2.2.5 foreach

Sintaxis

```
foreach ($matriz as $valor) {
    bloque_de_sentencias
}
```

El bucle foreach es muy útil para recorrer matrices cuyo tamaño se desconoce o matrices cuyos índices no son correlativos o numéricos (matrices asociativas).

El bucle ejecuta el bloque de sentencias tantas veces como elementos contenga la matriz $matriz y, en cada iteración, la variable $valor toma uno de los valores de la matriz.

Si deseamos acceder tanto al valor de la matriz como al índice:

Sintaxis

```
foreach ($matriz as $indice => $valor) {
    bloque_de_sentencias
}
```

El bucle ejecuta el bloque de sentencias tantas veces como elementos contenga la matriz $matriz y, en cada iteración, la variable $valor toma uno de los valores de la matriz y la variable $indice toma como valor el índice correspondiente.

Ejemplos:

```
<?php
$matriz = [0, 1, 10, 100, 1000];
foreach ($matriz as $valor) {
    print "<p>$valor</p>\n";
}
?>
```

```
<p>0</p>
<p>1</p>
<p>10</p>
<p>100</p>
<p>1000</p>
```

```
<?php
$matriz = ["red" => "rojo", "green" =>
"verde", "blue" => "azul"];
foreach ($matriz as $indice => $valor)
{
    print "<p>$indice : $valor</p>\n";
}
?>
```

```
<p>red : rojo</p>
<p>green : verde</p>
<p>blue : azul</p>
```

Prácticas

Los ejercicios con foreach los veremos después de conocer arrays