



Argentina Programa





Estructuras de control





Las estructuras de control son sentencias con las cuales podemos controlar el flujo de la información dentro de nuestro código, haciendo que éste se comporte de una manera determinada.



Entre las estructuras de control más comunes nosotros veremos las siguientes:



- If / else / elseif
- switch
- bucle while
- bucle do while
- bucle for
- bucle foreach





Estructura de control

IF (condicional)





```
1 <?php  
2  
3     $nombre = 'Carlos';  
4     $edad = 25;  
5  
6     if($edad > 18){  
7         echo $nombre . ' es mayor de edad';  
8     }  
9  
0 // Salida: Carlos es mayor de edad
```





```
6  if($edad > 30){  
7      echo $nombre . ' es mayor de edad';  
8  }  
9  
10 // Salida:
```





```
1 <?php  
2  
3 $nombre = "Pedro";  
4  
5  
6 if ($nombre == "Pedro") {  
7     echo "Su nombre es Pedro";  
8 }  
9  
10 // Salida: Su nombre es Pedro
```



else



existe una expresión que le indica al código qué hacer si la expresión dentro del IF no es verdadera:



else





```
1 <?php  
2  
3 $nombre = 'Carlos';  
4 $edad = 20;  
5  
6  
7 if($edad >= 18){  
8     echo $nombre . ' es mayor de edad';  
9 }else{  
10    echo $nombre . ' no es mayor de edad';  
11 }  
12  
13 // Salida: Carlos es mayor de edad
```



```
4  if( /* CONDICIÓN LÓGICA */){
5      // Ejecuta solo en caso de VERDADERO
6  }else{
7      // Ejecuta en el caso de FALSO
8 }
9
```



elseif



También nos puede pasar que queramos ejecutar código para ciertas condiciones lógicas, suponiendo que la primer condición del

IF
no se cumple, podemos volver a preguntar otra condición y si no se cumple ésta tampoco, ahí sí ejecutamos el
ELSE





```
1 <?php  
2  
3     $nombre = 'Carlos';  
4     $edad = 17;  
5  
6     if($edad > 18){  
7         echo $nombre . ' es un joven';  
8     }elseif($edad > 13){  
9         echo $nombre . ' es un adolescente';  
10    }else{  
11        echo $nombre . ' es un niño';  
12    }  
13  
14 // Salida: Carlos es un adolescente
```



```
1 <?php
2
3 $nombre = 'Carlos';
4 $edad = 42;
5
6 if($edad > 60){
7     echo $nombre . ' es un adulto mayor';
8 }elseif($edad > 40){
9     echo $nombre . ' es un adulto';
10 }elseif($edad > 18){
11     echo $nombre . ' es un joven';
12 }elseif($edad > 13){
13     echo $nombre . ' es un adolescente';
14 }else{
15     echo $nombre . ' es un niño';
16 }
17
18 // Salida: Carlos es un adulto
```





Estructura condicional **Switch**



Por buenas prácticas tenes que saber que no es bueno colocar tantos if anidados o condiciones consecutivas y es por eso que existe la función Switch que nos va a preguntar sobre el contenido de una variable y podemos designar para cada caso un comportamiento distinto.





```
1 <?php
2 $i = 2;
3
4 switch ($i) {
5     case 0:
6         echo "i es igual a 0";
7         break;
8     case 1:
9         echo "i es igual a 1";
10    break;
11    case 2:
12        echo "i es igual a 2";
13        break;
14 }
15
16 //Salida: i es igual a 2|
```



Estructura de control bucle While



Al igual que en todos los lenguajes de programación, el bucle while sigue una lógica de repetición que gira en torno a una condición. Esta condición marca cuando empieza, la duración y el final de la ejecución del bucle





```
1  <?php
2
3  $edad = 1;
4
5  while($edad < 10){
6
7      # Esto ejecutará 9 bucles, es decir,
8      # el código dentro del While se ejecutará 9 veces
9
10     $edad = $edad + 1;
11 }
12
```





Estructura de control bucle For



Al igual que en el while, el For es una estructura de control que nos permite hacer bucles, es decir, repeticiones hasta que una condición ya no se cumpla, pero en este caso el bucle For es un poco más específico ya que nos facilita ciertos comportamientos.





```
1 <?php  
2  
3 for($edad=1; $edad<10; $edad++){  
4     # Esto ejecutará 9 bucles, es decir,  
5     # el código dentro del For se ejecutará 9 veces  
6 }  
7
```





Este bloque de código se comporta EXACTAMENTE IGUAL al ejemplo hecho con el While.

```
1  <?php
2
3  $edad = 1;
4  while($edad < 10){
5      # Esto ejecutará 9 bucles, es decir,
6      # el código dentro del While se ejecutará 9 veces
7      $edad = $edad + 1;
8  }
9
10 for($edad=1; $edad<10; $edad++){
11     # Esto ejecutará 9 bucles, es decir,
12     # el código dentro del For se ejecutará 9 veces
13 }
```



Analicemos el For

```
for($edad = 1;      $edad < 10;      $edad++)
```

```
for(expresión1;      expresión2;      expresión3)
```

Analicemos el For

La primera expresión será la iniciadora, la que inicia nuestra condición a evaluar dentro del bucle... En este caso podemos pensar coloquialmente “okey arrancamos con edad = 1”.

```
for($edad = 1;      expresión2;      expresión3)
```

Analicemos el For

La segunda expresión podemos decir que es el limitante (o pensar que es la condición de nuestra expresión lógica), en este caso, decimos que el bucle se ejecutará hasta que edad sea menor a 10.

```
for($edad = 1;      $edad < 10;      expresión3)
```

Analicemos el For

Y por último, la tercera expresión nos indicará qué va a pasar con la variable \$edad al finalizar cada bucle. En este caso le indicamos \$edad++ (lo cual es exactamente lo mismo que decir \$edad = \$edad + 1) que hará que en cada bucle edad vaya aumentando de 1 en 1.

```
for($edad = 1; $edad < 10; $edad++)
```



```
1 <?php  
2  
3 ✓ for ($i = 0; $i <= 10; $i++){  
4     echo $i;  
5 }  
6  
7 // Salida: 012345678910  
8
```





Funciones



En PHP (y en la programación en general) hay algo llamado Funciones, que básicamente son instrucciones de código que nos permiten automatizar procesos (en ocasiones, procesos muy complejos) y de esta forma no estar repitiendo código continuamente.





Partes....

```
1  <?php  
2  
3  <?php  
4      function saludar(){  
5          echo "Hola, como estas?";  
6      }  
7      saludar();  
8  
9      # Salida: Hola, como estas?  
10  
11
```





Se declara la función saludar, la cual imprime un mensaje por pantalla.

Llamamos a la función por su nombre saludar();

A continuación, en el momento que se llama la función, podemos hacer de cuenta que se inserta el código que contiene la función.

Se muestra por pantalla el mensaje “Hola, como estas?”

```
1  <?php
2
3  function saludar(){
4      echo "Hola, como estas?";
5  }
6
7  saludar();
8
9  # Salida: Hola, como estas?
10
11
```





Parámetros de una función



Algo interesante que tienen las funciones es que podemos pasarle variables (o valores) para que haga algo con esos datos.





Supongamos que yo quiero automatizar el saludo y que según una lista de nombres, me salude a todas las personas de esa lista.

El código sería el siguiente:

```
<?php  
    ✓ function saludar($persona){  
        echo "Hola, como estas $persona?";  
    }  
    ✓  
    ✓
```

1 0 1
0 0 1
0 0 0



Implementación



```
1  <?php
2
3  ✓ function saludar($persona){
4      echo "Hola, como estas $persona?";
5  }
6
7  saludar("Juan");
8  saludar("Carlos");
9  saludar("Matias");
10
11 #Salida 1: Hola, como estas Juan?
12 #Salida 2: Hola, como estas Carlos?
13 #Salida 3: Hola, como estas Matias?
14
```





Parámetros por referencia



Un parámetro por referencia tiene por objetivo modificar el contenido de una variable que se le envía a una función.





```
1  <?php
2
3  ↘ function nuevoMensaje(&$texto){
4      $texto = 'hola cambiamos el mensaje';
5  }
6
7  $mensaje = 'un mensaje';
8  nuevoMensaje($mensaje);
9  echo $mensaje;
10
11 # Salida: hola cambiamos el mensaje
12
```



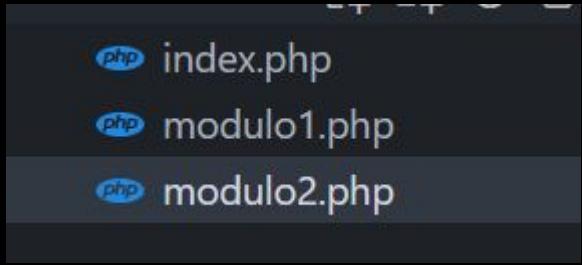
Modularizando un programa



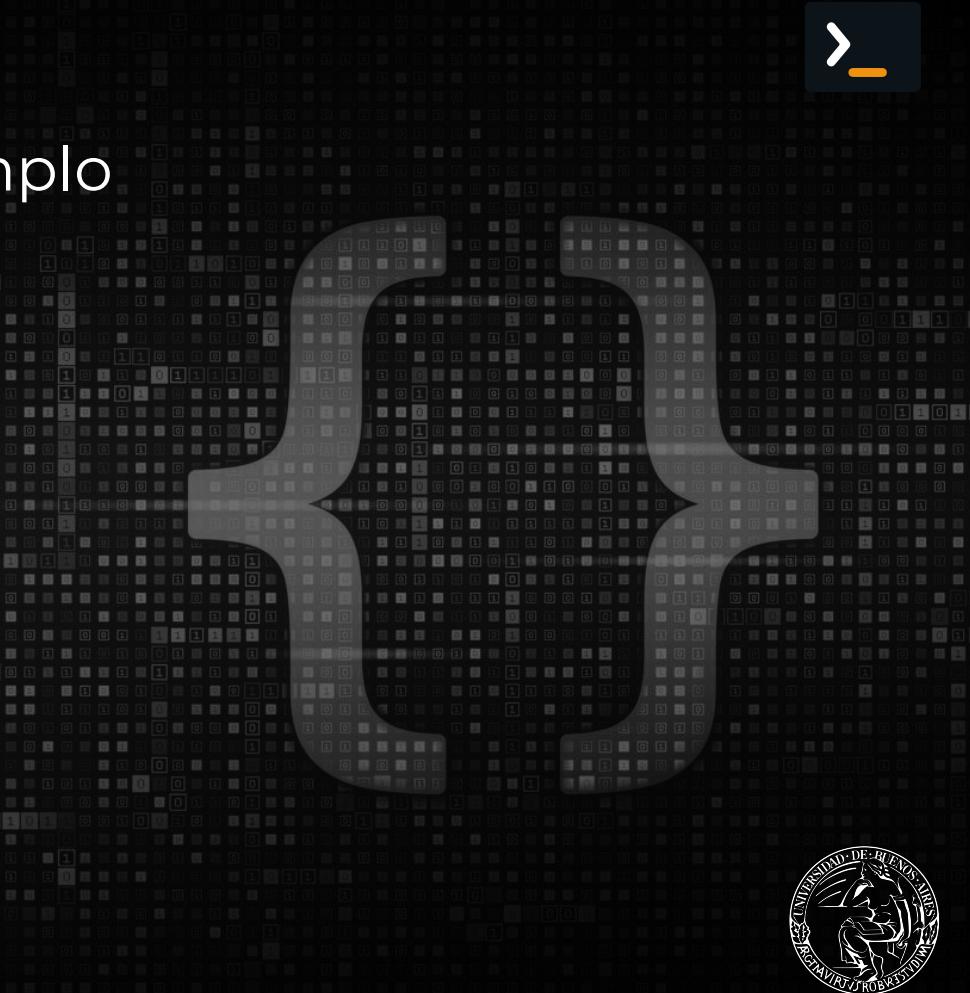
La idea de modularizar un programa es poder separar nuestro código en múltiples fragmentos (múltiples archivos) y al unirlos, trabajar como si estuviésemos trabajando todo en un solo archivo.



Miremos el siguiente ejemplo



Creamos 3 archivos PHP



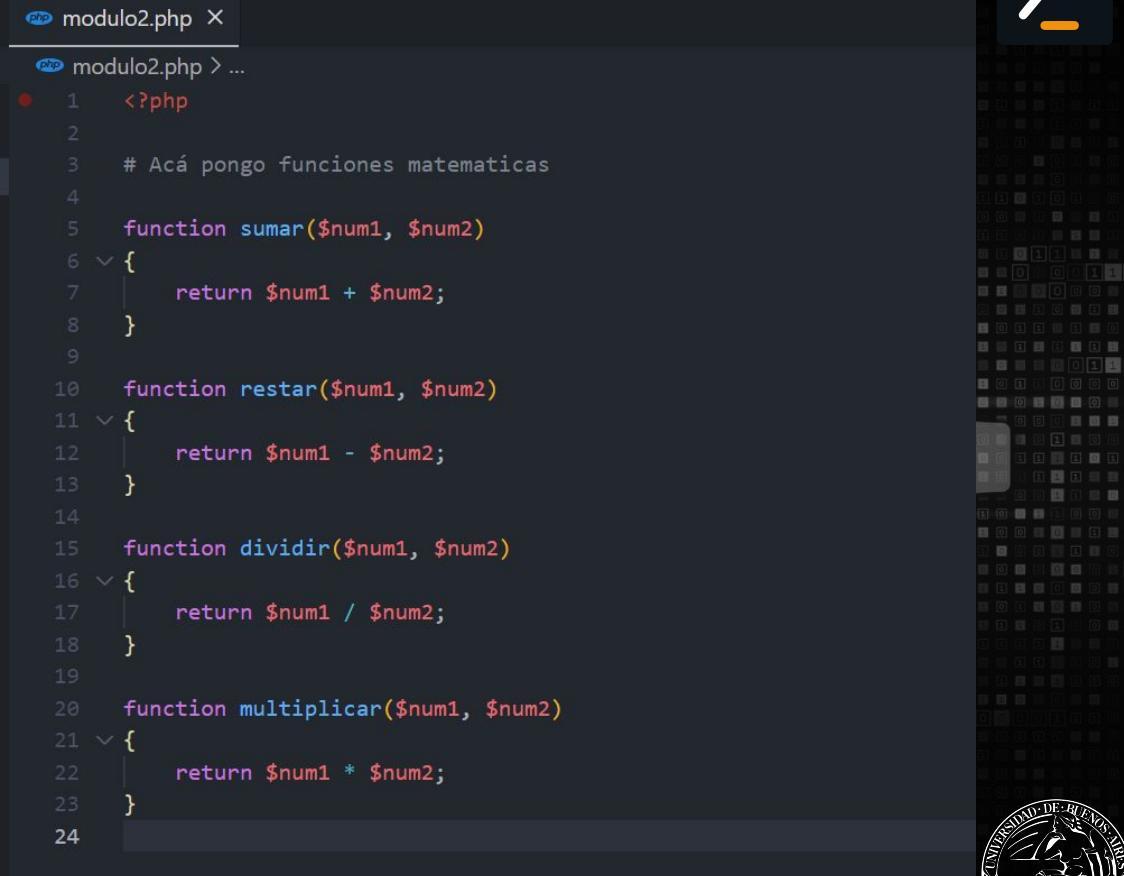


Primer archivo



```
php modulo1.php X
php modulo1.php > ...
1 <?php
2
3 function mostrar($variable){
4     echo $variable;
5 }
6
7 ✓ function mostrarTitulo($titulo){
8     echo "<h1>$titulo</h1>";
9 }
10
```

Segundo archivo



```
php modulo2.php X
php modulo2.php > ...
● 1  <?php
2
3  # Acá pongo funciones matematicas
4
5  function sumar($num1, $num2)
6  {
7      return $num1 + $num2;
8  }
9
10 function restar($num1, $num2)
11 {
12     return $num1 - $num2;
13 }
14
15 function dividir($num1, $num2)
16 {
17     return $num1 / $num2;
18 }
19
20 function multiplicar($num1, $num2)
21 {
22     return $num1 * $num2;
23 }
24
```



Al tercero lo llamaremos index y llamaremos desde allí a los otros 2 que creamos anteriormente



```
php index.php X
-----
php index.php > ...
1  <?php
2  include 'modulo1.php';
3  include 'modulo2.php';
4
5  $resultado = sumar(2604, 8521);
6
7  mostrar($resultado);
8
9
10
```



Salida

A screenshot of a web browser window titled "localhost/Cursos/PHP/". The main content area displays the number "11125".

localhost/Cursos/PHP/

localhost/Cursos/PHP/

11125



Formas de incluir archivos

php

index.php

```
1 <?php  
2  
3 include 'modulo1.php';  
4 include_once 'modulo1.php';  
5  
6 require 'modulo1.php';  
7 require_once 'modulo1.php';  
8
```



Implementación práctica

php index.php X

php index.php > html > body

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <title>Curso PHP</title>
5  </head>
6  <body>
7      <?php include 'titulo.php' ?>
8  </body>
9  </html>
```

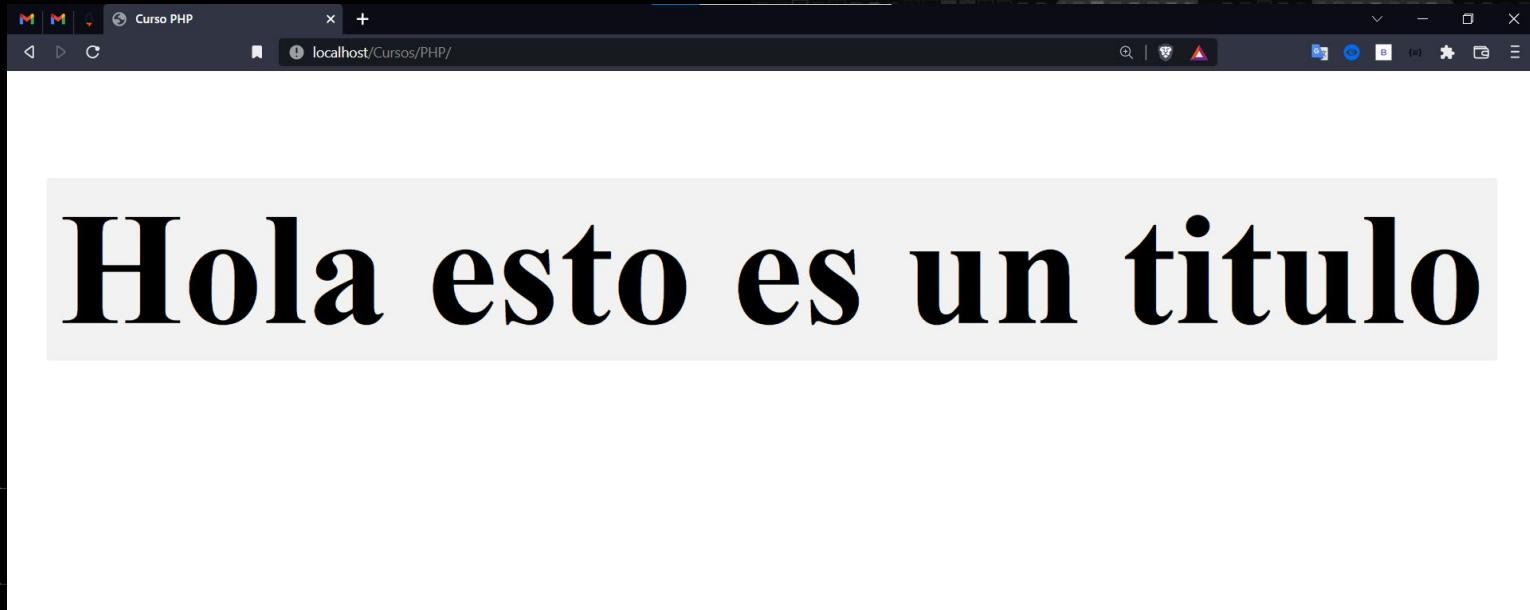
php titulo.php > ...

```
1  <div
2      style="width: 100%;
3      background: #f2f2f3;
4      text-align: center">
5
6      <h1>Hola esto es un titulo</h1>
7
8  </div>
```





Salida





Profundizando Arrays





Formas de declarar arreglos

index.php > ...

```
1  <?php  
2  
3      $myArray = array(['elemento0', 'elemento1', 'elemento2']);  
4      $myArray2 = ['elemento0', 'elemento1', 'elemento2'];  
5
```





index.php > ...

```
1  <?php  
2  
3      $myArray2 = [  
4          'elemento0',  
5          'elemento1',  
6          'elemento2'  
7      ];  
8  
9
```





php index.php > ...

```
1  <?php  
2  
3  $myArray = array([  
4      'elemento0',  
5      'elemento1',  
6      'elemento2'  
7  ]);  
8
```



Comprendiendo los arreglos

sueldos				
1200	750	820	550	490
sueldos[0]	sueldos[1]	sueldos[2]	sueldos[3]	sueldos[4]



Veámoslo en el código

```
index.php > ...
1  <?php
2
3  $sueldos = [
4      1200,
5      750,
6      820,
7      550,
8      490
9  ];
10
11 $sueldos[3];
12
13 # Salida: 550
14
```





iterando (recorriendo) arreglos

```
1  <?php
2
3  $sueldos = [
4  |    1200, 750, 820, 550, 490
5  ];
6  for($i=0; $i < count($sueldos); $i+1){
7  |    echo $sueldos[$i];
8  }
9
10 # Salida:   1200 750 820 550 490
11
```



Arreglos asociativos





php

index.php

```
1 <?php  
2  
3  
4 $sueldos['el_mayor'];  
5  
6  
7
```





Accediendo a arreglos mediante key

```
php index.php > ...
1 <?php
2
3 $sueldos = [
4     'el_mayor' => 1200,
5     'el_menor' => 490
6 ];
7
8 echo $sueldos['el_mayor'];
9
10 # Salida: 1200
11
```



Ejemplos

```
php index.php > ...
1  <?php
2  $ventas = [
3      'en_dia' => 1500,
4      'semanal' => 8300,
5      'mensual' => 142050
6  ];
7
8  echo $ventas['en_dia'];      # Salida:    1500
9  echo $ventas['semanal'];    # Salida:    8300
10 echo $ventas['mensual'];   # Salida: 142050
11
```





¿Qué pasa si tratamos de acceder a un índice que no existe?

```
1  <?php  
2  
3  
4  $array = [  
5      'Nombre' => 'Juan Carlos'  
6  ];  
7  
8  echo $array['Apellido'];  
9  
10 # Salida: ERROR - Index not found
```

