IAPWEB

TEMA 3 – PROGRAMACIÓN BÁSICA DE APLICACIONES CON PHP



EMPEZAR >

Powered by genially EDUCATION

< 00 >

Introducción

< 01 >

Bases de PHP

< 02 >

Variables

< 03 >

Estructuras de control

< 04 >

Formularios

< 05 >

Ejercicios

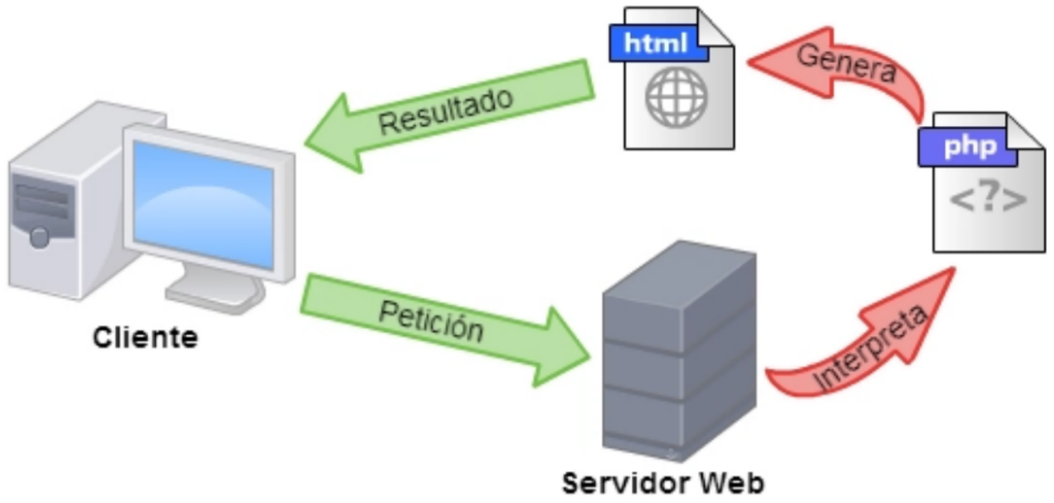


Powered by genially EDUCATION

Introducción a PHP

Qué es PHP:
PHP (acrónimo de **PHP: Hypertext Preprocessor**) es un lenguaje de programación de **código abierto** muy popular especialmente adecuado para el desarrollo web y que puede ser incrustado dentro de código HTML.
Es el lenguaje más popular para crear contenidos web en el lado del servidor (**lenguaje de script de servidor**).

Cómo funciona PHP:
El usuario escribe una URL en su navegador y éste envía un petición al **servidor web**. El servidor lee la página PHP (la interpreta) y envía al navegador del usuario un documento compilado en formato HTML.



Powered by genially EDUCATION

VENTAJAS DE PHP

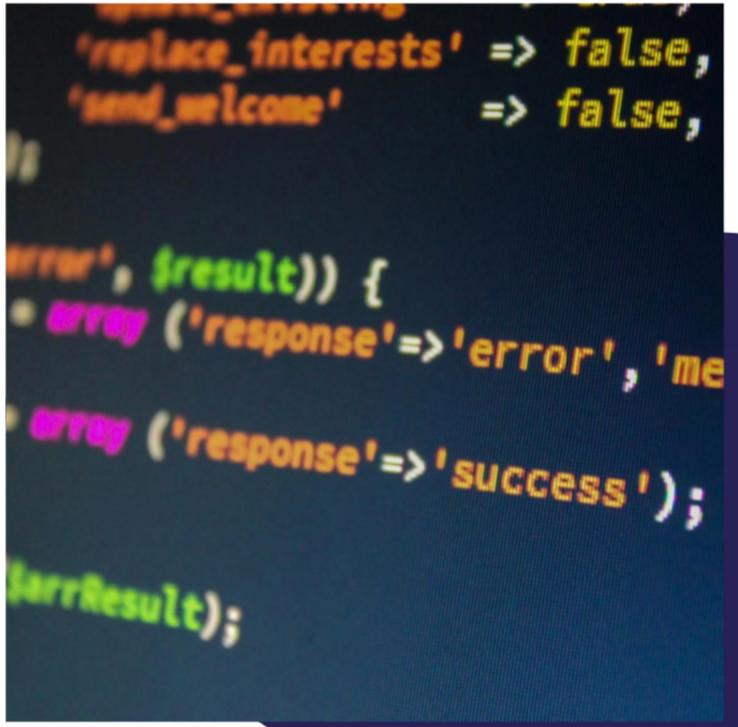
1. Multiplataforma
2. Abierto y gratuito
3. Gran comunidad de usuarios
4. Gran relación y compatibilidad con Apache y MySQL
5. LAMP, WAMP y XAMP
6. Gran cantidad de extensiones
7. Fácil (tipos de datos poco restrictivos y ausencia de reglas rígidas)
8. Múltiples editores de texto



Powered by genially EDUCATION

HERRAMIENTAS PARA LA ESCRITURA DE APLICACIONES EN PHP

1. **Bloc de notas / nano / vi**
2. **Editores de código multipropósito:** especializados en crear código fuente para diversos lenguajes, que se adaptan al lenguaje utilizado. Por ejemplo, **Notepad++**
3. **Entornos de desarrollo integrado (IDE):** software que incluye todo lo necesario para escribir aplicaciones PHP: coloreado del código, corrección automática, ayuda con la sintaxis, depuración, integración con sistemas de control de versiones, etc. Por ejemplo, **Eclipse, Netbeans, Visual Studio Code,** etc.
4. **Herramientas de desarrollo visual:** software que permite crear documentos web dibujando. Por ejemplo, **Adobe Dreamweaver**

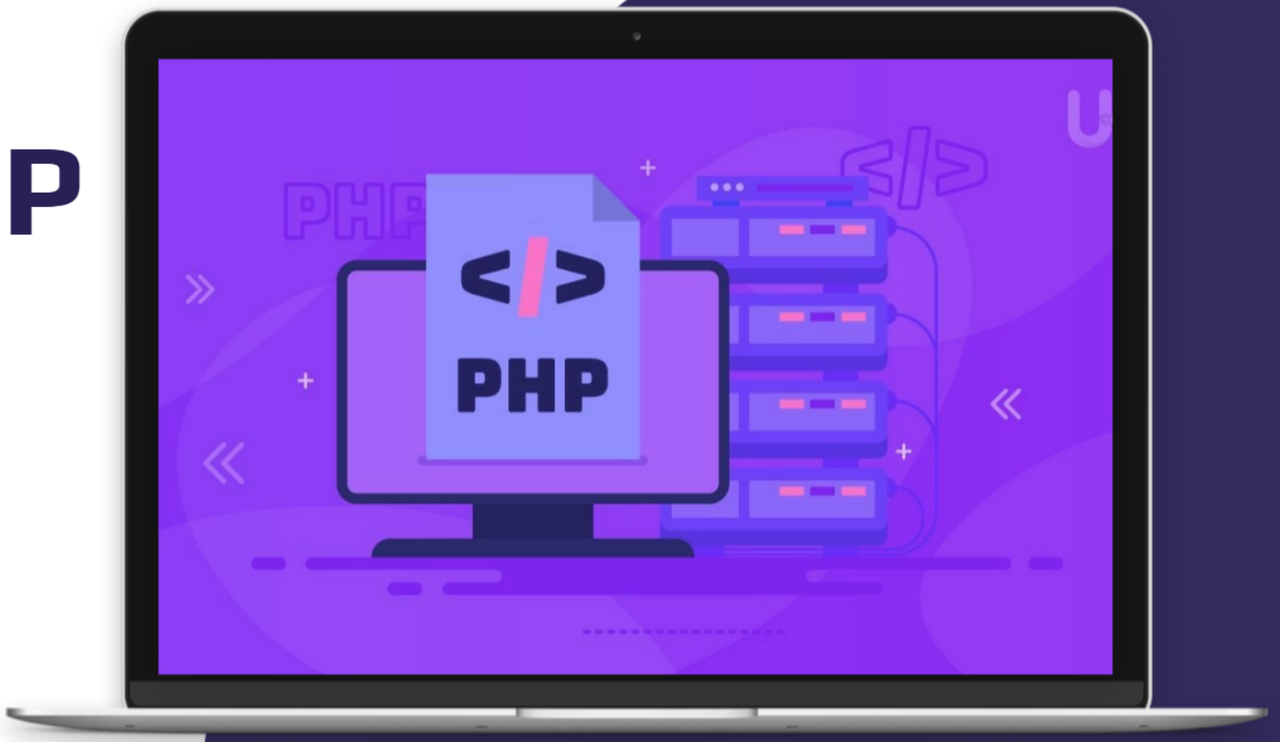


Powered by genially EDUCATION

<01>
BASES DE PHP

Primeros pasos

EMPEZAR >



Powered by genially EDUCATION

El código PHP se incrusta en un documento HTML, al que habrá que cambiarle la extensión **.html** por **.php.**
La manera de incrustarlo es la siguiente:

Powered by genially EDUCATION



¿Y por qué hace eso el código anterior?

La función **echo** se utiliza para mostrar un texto por pantalla. Este texto, cuando es un literal (no un número), debe ir entre comillas.

Se pueden encadenar textos utilizando la coma (,) o el punto (.):

Powered by genially EDUCATION

NORMAS BÁSICAS PARA ESCRIBIR EN PHP

- Todas las líneas de código deben finalizar con un punto y coma (**;**).
- Se puede agrupar el código en bloques que se escriben entre llaves (**{ }**).
- Una línea de código se puede partir o sangrar (añadir espacios al inicio) a voluntad con el fin de que sea más legible, siempre y cuando no partamos una palabra o un valor.
- PHP obliga a ser estrictos con las mayúsculas y minúsculas en algunos casos, como en el nombre de las variables. Sin embargo, con las palabras reservadas del lenguaje, no es estricto. Es ecir, para PHP los términos **WHILE**, **while** e incluso **wHiLe** son sinónimos. Sin embargo **$variable** y **$VARIABLE** no, al ser nombres de variables.
  El consejo sobre esta norma es utilizar siempre minúsculas, salvo para dar nombre a las constantes y separar palabras en nombres de variables o funciones, como veremos más adelante.

Powered by genially EDUCATION

COMENTARIOS

Es muy importante al programar añadir información en nuestro código que explique qué es lo que hace. Cuando tengamos que modificar un programa de 10000 líneas, agradeceremos esa ayuda.
Tenemos 3 tipos de comentarios:

- **Comentarios de varias líneas.** Comienzan con /* y terminan por */. Afectan a todo lo que escribamos entre esos dos símbolos.
- **Comentarios de una línea, estilo C++.** Se ponen tras la barra doble (//). Solo afectan a la línea en la que se colocan.
- **Comentarios de una línea, estilo ShellScript.** Se ponen tras la almohadilla (#). Solo afectan a la línea en la que se colocan.

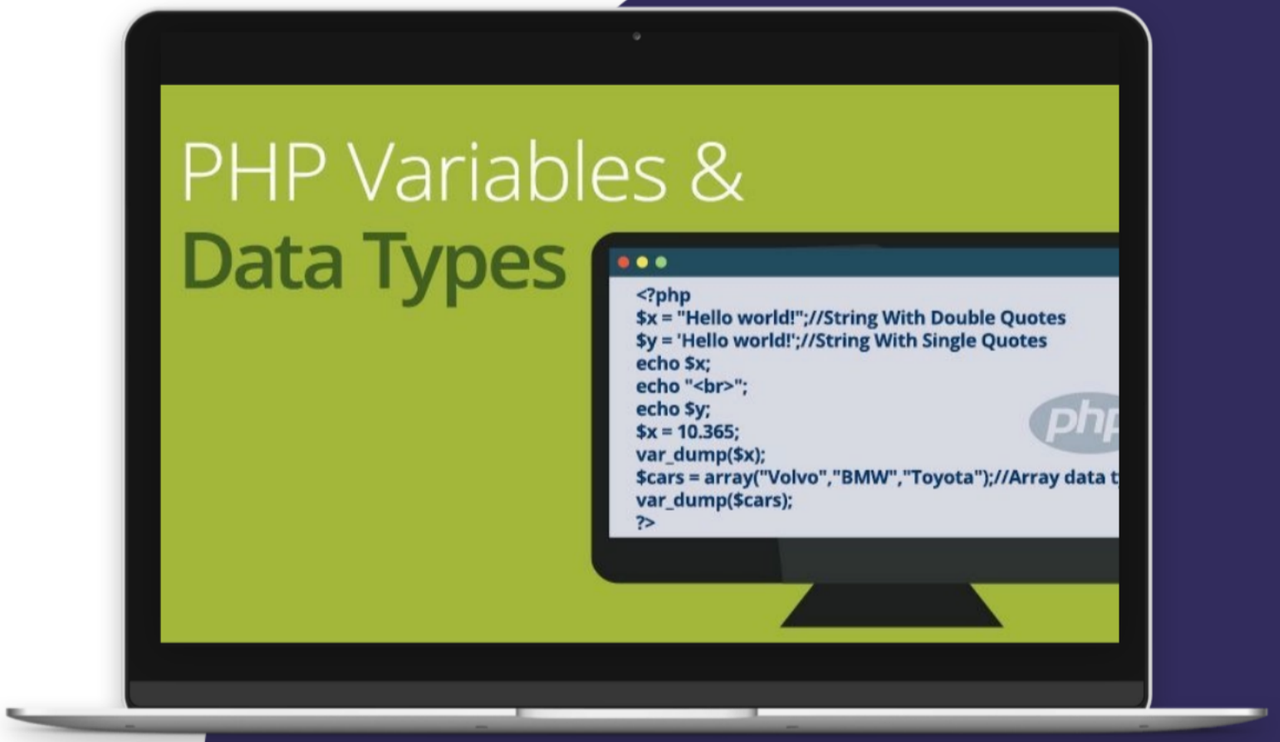
```
/*
Calcula el precio de la entrada en base a su edad, si
tiene carnet y el día de la semana
El precio final no puede ser inferior a 5 euros
*/
function calcularPrecioFinal($precioInicial, $edad,
$carnet, $dia){
    //Creamos una variable para ir actualizando el precio
    $precio = $precioInicial;
    if ($edad > 65){
        //Aplicamos un 20% de descuento
        $precio *= 0.8;
    }
    if ($carnet){
        //Aplicamos una rebaja de 2€
        $precio -= 2;
    }
```

Powered by genially EDUCATION

<02>
VARIABLES

¿Qué son y para qué sirven?

EMPEZAR >



Powered by genially EDUCATION

INTRODUCCIÓN A LAS VARIABLES

En todos los lenguajes de programación, las variables son contenedores que sirven para almacenar los datos que utiliza un programa. Es decir, son los nombres que asociamos a determinados datos.
La realidad física es que cada variable ocupa un espacio en la memoria RAM del servidor. Es decir, cuando utilizamos el nombre de la variable realmente estamos haciendo referencia a un dato que está en memoria.
El nombre de las variables se conoce como **identificador** de variable. En PHP, un identificador tiene que cumplir las siguientes reglas:

Powered by genially EDUCATION

DECLARAR VARIABLES Y ASIGNAR VALORES

A diferencia de otros lenguajes de programación, en PHP no es necesario declarar una variable; se utiliza y ya está. Es decir, si queremos que la variable **$edad** valga 15, haremos:

```
$edad=15;
```

Fíjate que para asignarle un valor, utilizamos el símbolo "=".
En el caso de asignar números se escriben tal cual, los decimales se escriben utilizando el punto (.) y los textos se encierran entre comillas simples o dobles.

```
$nombre="Anselmo";      $precio=12.95;
```

También se pueden asignar resultados de aplicar fórmulas mediante operadores, como por ejemplo:

```
$beneficios = ($gastos - $ingresos) * 0.6;
```

En PHP, no es necesario indicar de qué tipo es la variable, lo cual permite cambiar el tipo de datos que almacena. Así, este código es perfectamente válido:

```
$x=18; //asignamos un número
$x="Hola"; //asignamos un texto
```

Powered by genially EDUCATION

¿Y QUÉ PASA SI A LA VARIABLE NO LE ASIGNAMOS NINGÚN VALOR?

Si intentamos utilizar una variable a la que no se le ha asignado ningún valor, nos saldrá un error similar a este: **Warning**: Undefined variable $x in **C:\xampp\htdocs\test\test.php** on line **10**

Si quisiéramos modificar los errores de aviso que se lanzan para que solo avise si hay errores de nivel usuario (o más graves), tendríamos que modificar la directiva **error_reporting** del archivo **php.ini**, asignándole el valor **E_USER_ERROR**.

En nuestro caso, lo dejaremos como está, porque nos interesa ver todos los errores.

```php
<?php
    echo $x; //Es la primera vez que se usa $x. Error
    $y = $x + 2; //Error. Se usa $x sin haberla definido
    echo $y; //$y valdrá 2
?>
```

Powered by genially EDUCATION

TIPOS DE DATOS

- NÚMEROS ENTEROS
- NÚMEROS EN COMA FLOTANTE
- CADENAS O STRINGS
- BOOLEANOS
- CONVERSIONES ENTRE TIPOS

Powered by genially EDUCATION

CONSTANTES

Las constantes almacenan valores que no cambian en el tiempo.
Es conveniente nombrarlas en mayúsculas y se pueden definir de dos formas:

```
define("PI", 3.141592);
```

```
const PI=3.141592;
```

Las constantes no utilizan el signo **$** de las variables, simplemente utilizan el nombre. Es decir, para mostrar el valor de PI usaríamos:

```
echo PI;
```

Powered by genially EDUCATION



OPERADORES

- RELACIONALES
- LÓGICOS
- ARITMÉTICOS
- DE ASIGNACIÓN
- DE BIT
- CONCATENACIÓN

Powered by genially EDUCATION

VAMOS A PONER EN PRÁCTICA LO APRENDIDO

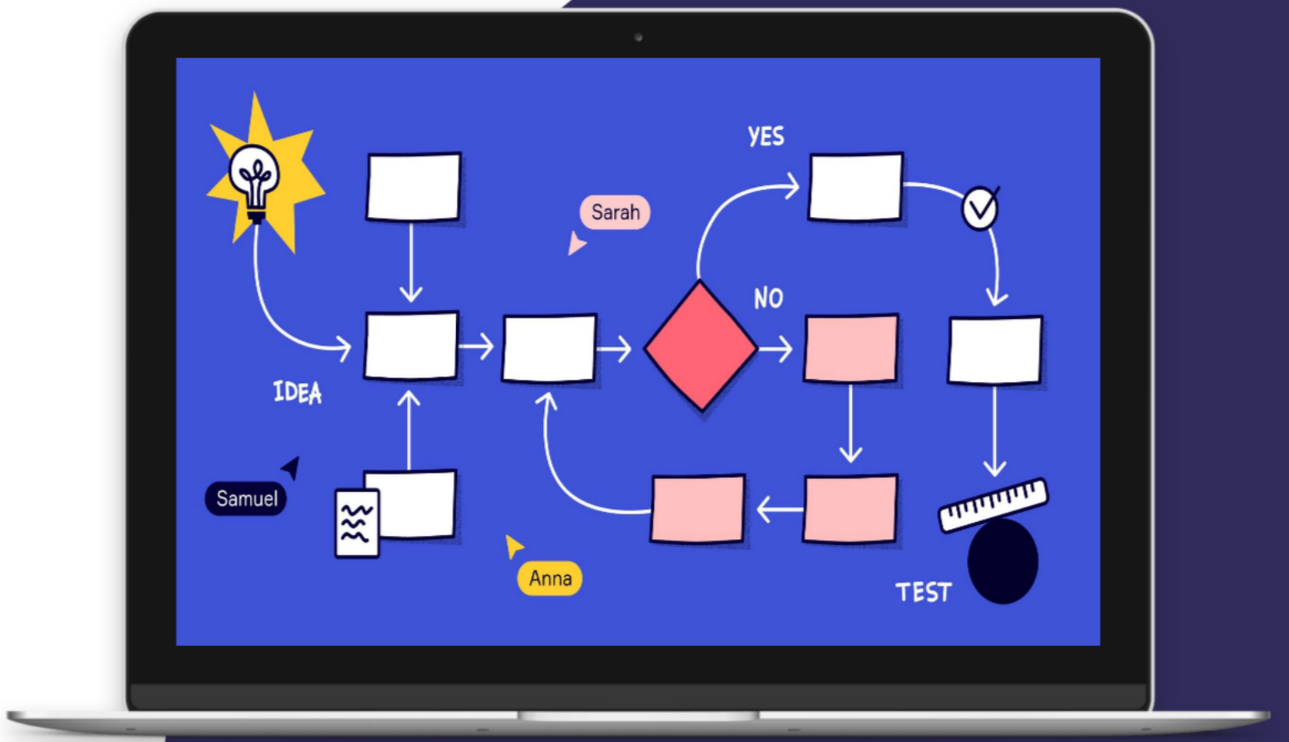
EJERCICIOS >

Powered by genially EDUCATION

<03> ESTRUCTURAS DE CONTROL

Vamos a complicarlo un poco

EMPEZAR >



Powered by genially EDUCATION



ESTRUCTURAS DE CONTROL

Hasta ahora, las instrucciones que hemos visto se ejecutan secuancialmente; es decir, podemos saber lo que hace el programa leyendo las líneas de izquierda a derecha y de arriba abajo.
Las estructuras de control de flujo permiten alterar esta forma de ejecución. A partir de ahora, habrá líneas en el código que se ejecutarán o no (o que se ejecutarán varias veces) dependiendo de una **condición**. Esa condición se construye utilizando lo que se conoce como **expresión lógica**; esto es, cualquier tipo de expresión que retorne un resultado verdadero o falso.
Las expresiones lógicas se construyen a través de los operadores relacionales y lógicos vistos anteriormente.
Existen dos tipos de estructuras: **sentencias condicionales** y **bucles**.

SENTENCIAS CONDICIONALES>

BUCLES>

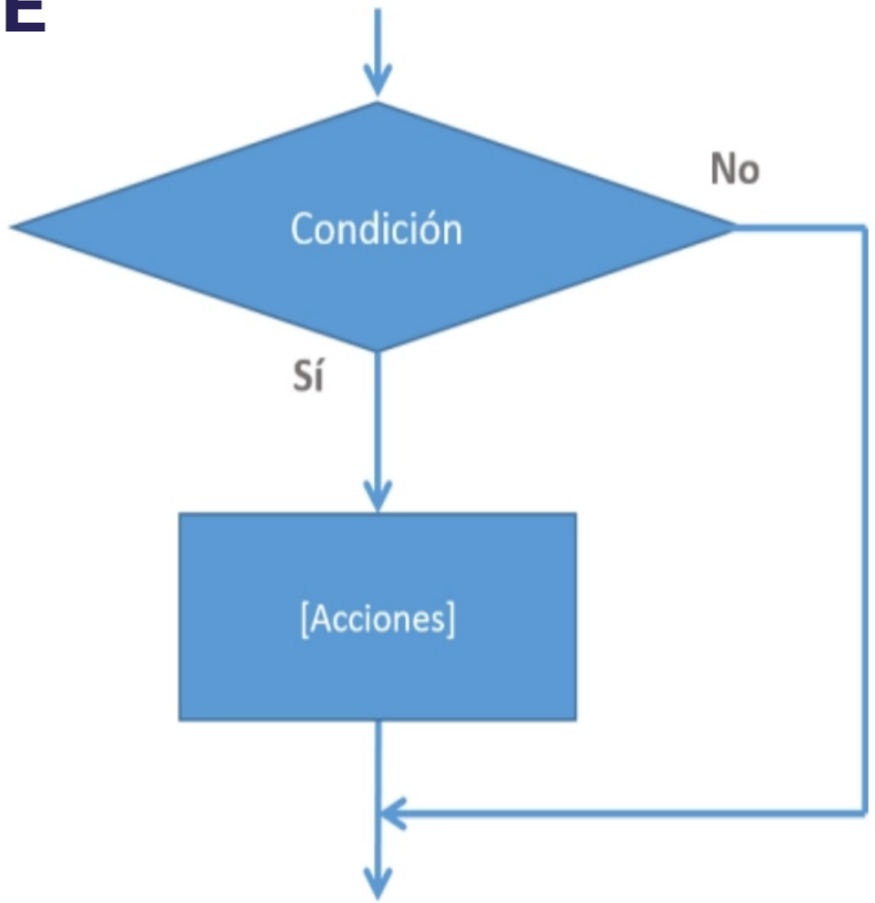
Powered by genially EDUCATION

SENTENCIA CONDICIONAL IF SIMPLE

Nos permite ejecutar un bloque de código si se cumple una determinada condición. Su estructura es:

```
if (condición) {
   instrucciones;
}
```

Las llaves solo son obligatorias si va a haber varias instrucciones. Si no, podemos obviarlas.



Ejemplo de uso:

```
if ($nota >= 5){
      echo "Aprobado";
      $numeroDeAprobados++;
}
```

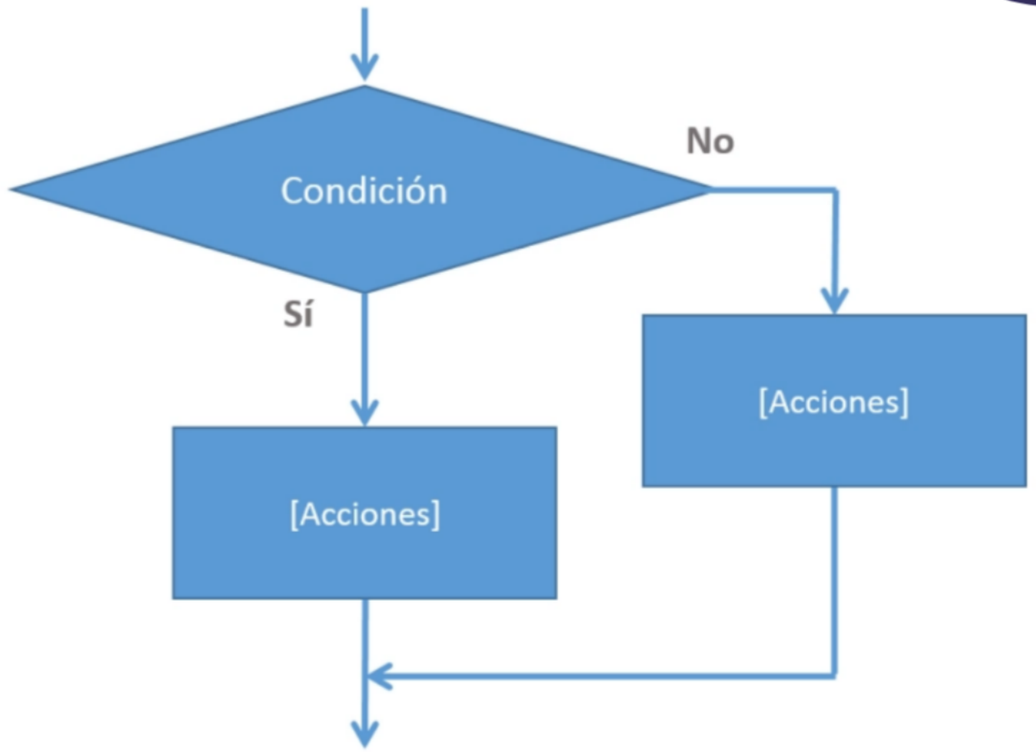
Powered by genially EDUCATION

SENTENCIA CONDICIONAL IF COMPUESTA

Nos permite ejecutar un bloque de código alternativo si la condición del if no se cumple. Su estructura es:

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

```
if ($nota >= 5){
      echo "Aprobado";
      $numeroDeAprobados++;
}
else{
      echo "Suspenso";
      $numeroDeSuspensos++;
}
```



Powered by genially EDUCATION

SENTENCIA CONDICIONAL IF MÚLTIPLE

Nos permite evaluar múltiples condiciones en secuencia. Su estructura es:

```
if (condición1) {
   // código a ejecutar si la condición1 es verdadera
} elseif (condición2) {
   // código a ejecutar si la condición2 es verdadera
} else {
   // código a ejecutar si ninguna condición es verdadera
}
```

```
if ($nota < 5){
      echo "Suspenso";
      $suspensos++;
}
elseif ($nota < 7){
      echo "Aprobado";
      $aprobados++;
}
elseif ($nota < 9){
      echo "Notable";
      $aprobados++;
}
else{
      echo "Sobresaliente";
      $aprobados++;
}
```

Powered by genially EDUCATION

SENTENCIA SWITCH

Es una alternativa a la sentencia condicional múltiple. 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.



Su estructura es:

```
switch (expresión) {
    case valor1:
        instrucciones del valor 1
        [break]
    [case valor 2:
        instrucciones del valor 2
        [break]
    ]
    [...]
    [default:
        instrucciones que se ejecutan en cualquier
        otro caso
    ]
}
```

(Todo lo que está entre corchetes es opcional)

Powered by genially EDUCATION

SWITCH - EJEMPLOS

```
switch ($diaSemana){
    case 1:
        $dia="Lunes";
        break;
    case 2:
        $dia="Martes";
        break;
    ...
    case 7:
        $dia="Domingo";
        break;
    default:
        $dia="?";
}
```

No usar **break** para salir del switch puede permitir crear estructuras switch que ejecuten unas mismas instrucciones para diferentes valores:

```
switch ($diaSemana){
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
        echo "Laborable";
        break;
    case 6:
    case 7:
        echo "Festivo";
        break;
    default:
        echo "?";
}
```

Powered by genially EDUCATION

VAMOS A PONER EN PRÁCTICA LO APRENDIDO

EJERCICIOS >

Powered by genially EDUCATION

BUCLE WHILE

Es una estructura que permite que un conjunto de sentencias se repitan mientras se cumpla una determinada condición.

Su sintaxis es la siguiente:

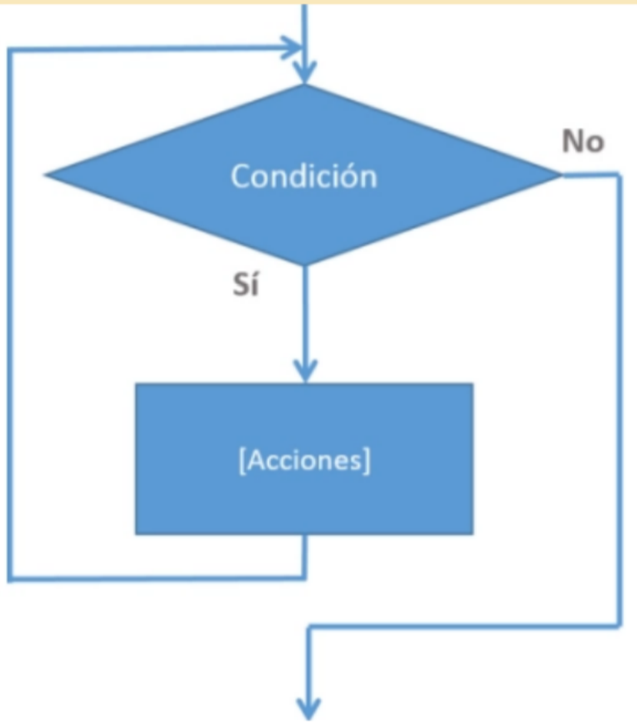
```
while (expresiónLógica){
    sentencias que se ejecutan si expresiónLógica es true
}
```

Un bucle while sigue estos pasos:

1. Se evalúa la expresión lógica
2. Si la expresión es verdadera, se ejecutan las sentencias. Si no, el programa abandona la sentencia while
3. Tras ejecutar las sentencias, volvemos al paso 1

Ejemplo, bucle que escribe números del 1 al 100:

```
$i=1;
while ($i<=100){
    echo $i."<br/>";
    $i++;
}
```



Powered by genially EDUCATION

TIPOS DE WHILE

Dentro de los bucles while, distinguimos dos tipos: los de **contador** y los de **centinela**. Los bucles de contador se repiten un número determinado de veces. El contador es una variable que va modificando su valor en cada vuelta del bucle. Cuando el contador alcanza un valor determinado, termina.

El contador es $i, que se va incrementando en 1 en cada iteración hasta que pasa a valer más de 100

```
$i=1;
while ($i<=100){
    echo $i."<br/>";
    $i++;
}
```

Powered by genially EDUCATION

BUCLE DO-WHILE

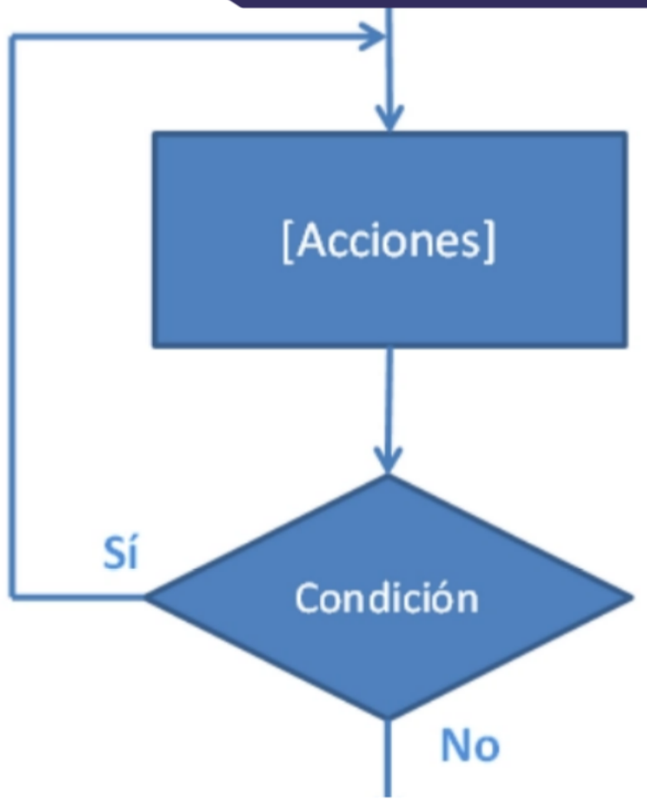
La única diferencia respecto a los bucles while es que la expresión lógica se evalúa después de haber ejecutado las sentencias una vez.

Su sintaxis es la siguiente:

```
do {
      sentencias
} while (expresiónLógica)
```

Ejemplo, bucle que escribe números del 1 al 100:

```
$i=1;
do {
      echo $i."<br/>";
      $i++;
} while ($i<=100)
```



Powered by genially EDUCATION

BUCLE FOR

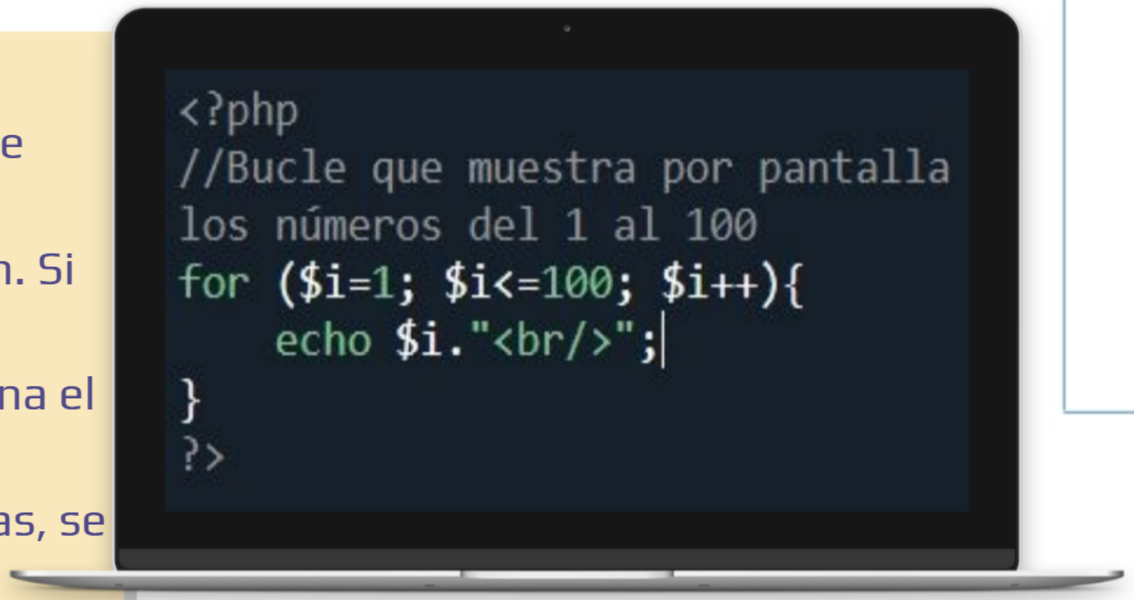
Es el bucle más utilizado por ser más cómodo y menos propenso a errores, pensado especialmente para implementar bucles de contador.

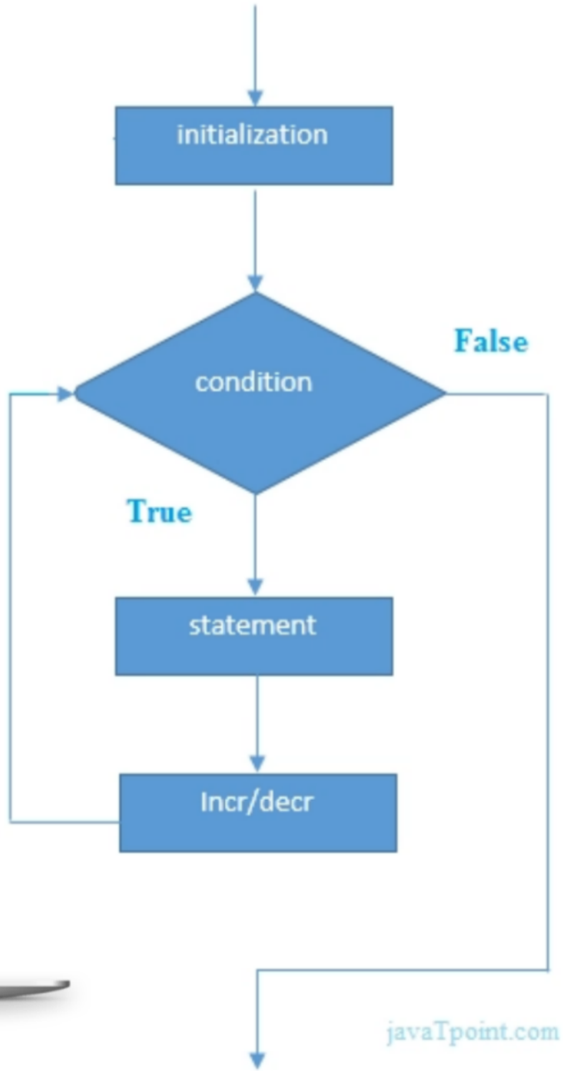
Su sintaxis es la siguiente:

for (inicialización; condición; incremento){
sentencias
}

Su funcionamiento es:

1. Se ejecuta la instrucción de inicialización
2. Se comprueba la condición. Si es cierta, se ejecutan las sentencias. Si no, se abandona el bucle
3. Tras ejecutar las sentencias, se ejecuta la instrucción de incremento y volvemos al paso 2





Powered by genially EDUCATION

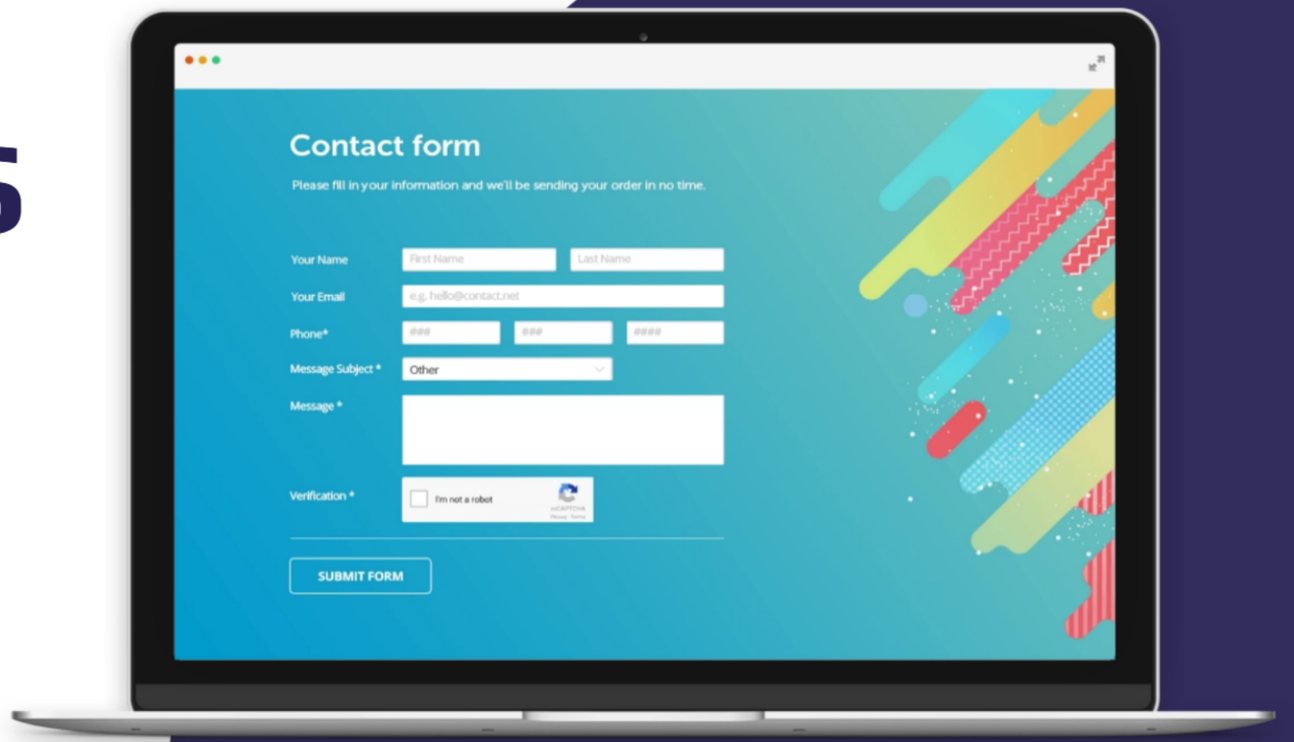
VAMOS A PONER EN PRÁCTICA LO APRENDIDO

EJERCICIOS >

Powered by genially EDUCATION

<04>
USO DE FORMULARIOS HTML DESDE PHP

EMPEZAR >



Powered by genially EDUCATION

ENVÍO DE DATOS DESDE UN FORMULARIO

Los formularios son los elementos de las páginas web que permiten recoger valores del usuario y enviarlos a una dirección URL para procesarlos.
HTML dispone de numerosos controles para almacenar la información que se pide al usuario: cuadros de texto, botones de radio, etc.
Se debe tener en cuenta que:

- Todos los controles del formulario deben estar dentro de un elemento **form**, que tendrá los siguientes atributos:
  - **method**: se refiere a la forma en la que se pasan los datos del formulario. Puede ser **get** o **post**.
  - **action**: indica la URL a la dirección del servicio que recoge los datos.
- Dentro del elemento form se colocan los elementos HTML capaces de recoger información (**input**, **option**, **textarea**, etc). Estos poseen un atributo **name**, en el que se indica el nombre de la variable que se envía al servidor; y otro **value**, que permite indicar el valor que se le asigna inicialmente a esa variable.

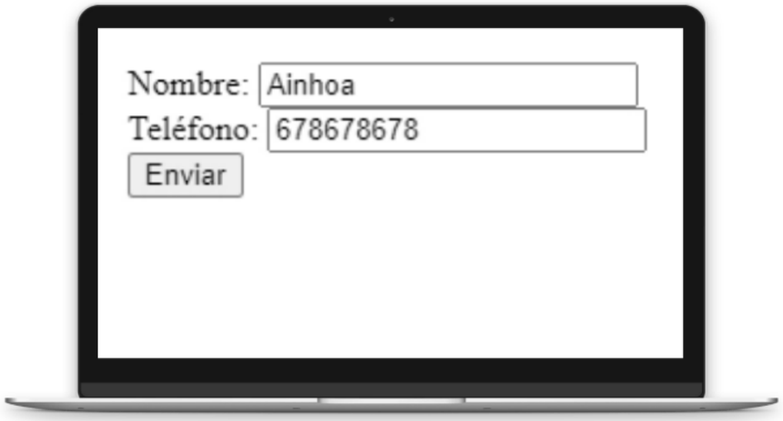
Powered by genially EDUCATION

PASO DE DATOS MEDIANTE GET

El método GET lo que hace es añadir a la URL destinataria del formulario los nombres y valores recogidos en el formulario.

Ejemplo: tenemos el siguiente formulario y lo rellenamos con los siguientes datos:

```
<form action="recogida.php"
method="get">
    <label for="nombre">Nombre:</label>
    <input type="text" name="nombre"/>
<br/>
    <label for="telefono">Teléfono:
</label>
    <input type="text" name="telefono"
maxlength="9" pattern="[0-9]{9}"/><br/>
    <input type="submit"
value="Enviar"/><br/>
</form>
```



Al pulsar enviar, se genera la URL **http://localhost/recogida.php?nombre=Ainhoa&telefono=678678678**
Es decir, a la URL original le añade pares variable-valor.

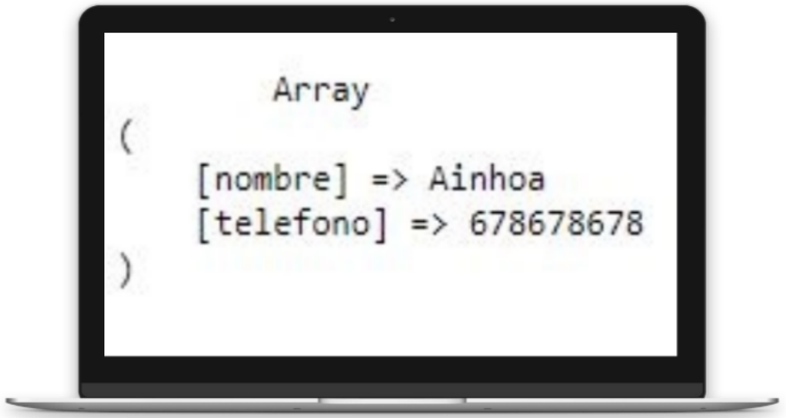
Powered by genially EDUCATION

PASO DE DATOS MEDIANTE GET

Supongamos que la página receptora, en la dirección **http://localhost/recogida.php** tiene el siguiente código:



Esta página escribiría



Es decir, un array que contiene cada elemento pasado desde el formulario. Cada elemento tiene como clave el nombre y como valor, el valor que el formulario asocia a ese nombre. Acceder a un valor concreto del array se haría, por ejemplo, mediante **$_GET["telefono"]**.

Si intentamos acceder mediante **$_GET["nombre"]** al nombre de un valor que no hemos recibido (por ejemplo el apellido), obtendremos un error. Por eso, siempre antes de acceder a esa variable debemos comprobar si existe y, además, no está vacía. Esto lo haremos mediante la instrucción:

if(isset($_GET["apellido"]) && !empty($_GET["apellido"]))

Powered by genially EDUCATION

PASO DE DATOS MEDIANTE POST

Los formularios se crean igual en ambos casos pero ahora cambiamos el atributo **method** de la etiqueta **form** para que tome el valor **post**.

Mediante POST, los nombres y valores del formulario no viajan en la URL, sino que se colocan en la cabecera del paquete http; por lo que los dotamos de una menor visibilidad.

Para acceder a los valores pasados mediante POST, utilizamos el array **$_POST** de la misma manera que hicimos antes con el **$_GET**. Por ejemplo, si queremos acceder al campo "teléfono", utilizaremos **$_POST["telefono"]**.

Al igual que con el GET, siempre debemos comprobar antes si hemos recibido ese valor del formulario mediante las función **isset** y **empty**.

A veces, queremos usar la misma página para el formulario y la recepción.
Ver un ejemplo

También, podemos redirigir al usuario a otra página mediante la función **header**.
header("Location:http://www.google.es");

Powered by genially EDUCATION

VARIABLES OCULTAS

Imagina que tenemos que pasar unos datos no solo de un formulario a la página que los recoge, sino a otras páginas, pero no queremos que el usuario tenga que introducir esos datos de nuevo. Esto lo haríamos utilizando en un formulario un **input de tipo hidden**.
Veamos un ejemplo:

Powered by genially EDUCATION

VARIABLES OCULTAS

```php
<!-- Recogemos el nombre, saludamos y en un nuevo formulario
 pedimos los apellidos y redirigimos a otra página -->
    <?php
        $nombre = $_GET["nombre"];
        echo "Hola, $nombre, introduce tus apellidos.";
    ?>
    <form action="recogida.php" method="get">
        <label for="apellidos">Apellidos:</label>
        <input type="text" name="apellidos"/>
        </br>
        <input type="submit" value="Enviar"/></br>
    </form>
```



Hola, Carmen, introduce tus apellidos.
Apellidos:
Enviar

Powered by genially EDUCATION

VARIABLES OCULTAS

La solución pasa por pasar el nombre como input oculto en el segundo formulario:

```
<form action="recogida.php" method="get">
    <!-- Pasamos el nombre como un campo oculto
     para que se haga de manera transparente al usuario -->
    <input type="hidden" name="nombre" value="<?php echo $nombre ?>"/>

    <label for="apellidos">Apellidos:</label>
    <input type="text" name="apellidos"/>
    </br>
    <input type="submit" value="Enviar"/></br>
</form>
```



Hola, Carmen Sánchez.

Powered by genially EDUCATION

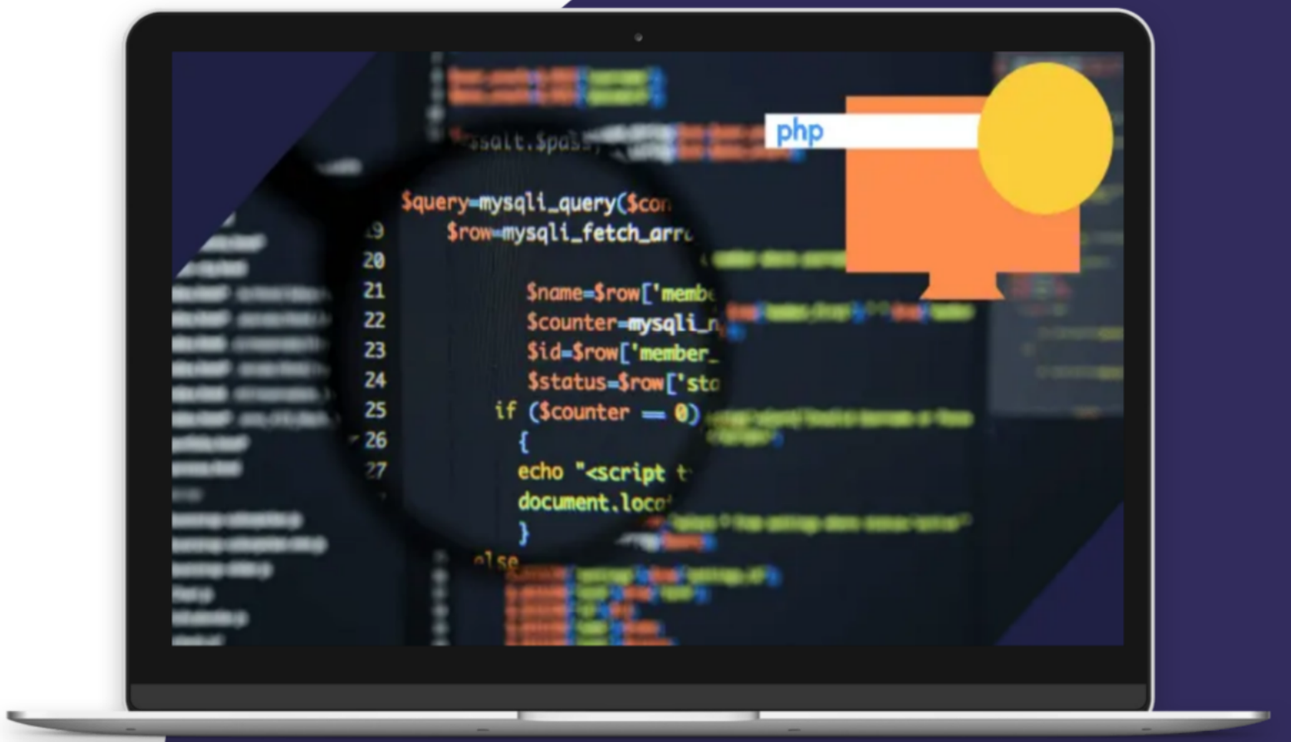
VAMOS A PONER EN PRÁCTICA LO APRENDIDO

EJERCICIOS >

Powered by genially EDUCATION

<05>
EJERCICIOS

EMPEZAR >



Powered by genially EDUCATION

EJERCICIOS

ENUNCIADOS

EJERCICIOS RESUELTOS

Powered by genially EDUCATION

Tema 3 finalizado



Powered by genially EDUCATION