

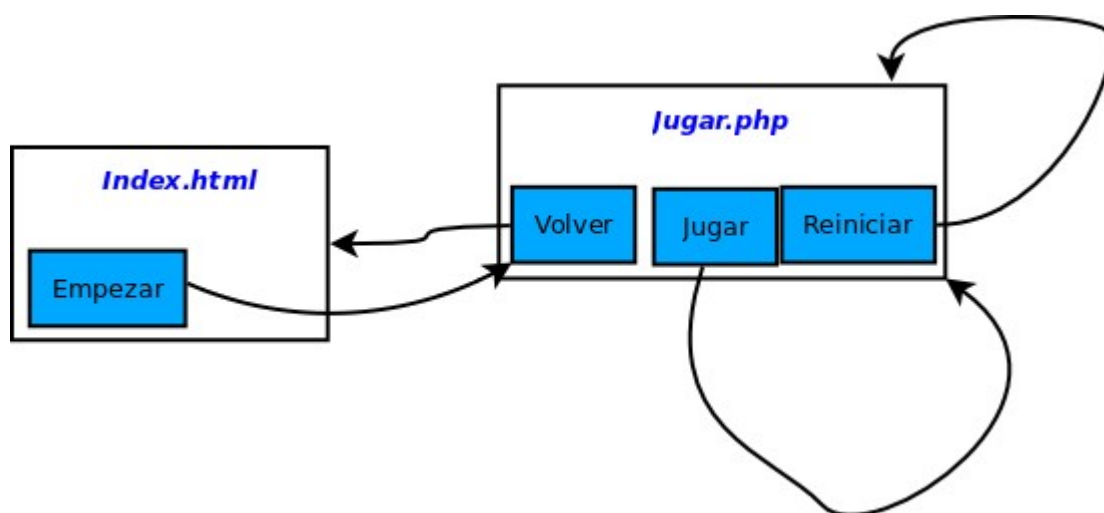
Recomendaciones de buenas prácticas de programación que observo que no se han tenido en cuenta en general

Este apartado lo considero muy importante y lo tendré en cuenta en futuras prácticas que se cumplan. Tomadlo como si fuera teoría importante. Intentaré a cada una en vuestra corrección de la práctica referenciároslo respecto a vuestro código

Buenas prácticas 1

Si tengo diferentes opciones (botones submit) que me pueden llevar a una página una forma buena de hacerlo es ponerles a todos los submit el mismo name. Tomemos como ejemplo esta página

En este caso el html



index.php

```
<form action="jugar.php" method="POST">
.....
<input type="submit" value="Empezar" name="submit">
</form>
```

En jugar.php

```
<form action="jugar.php" method="POST" >
.....
<input type="submit" value="Jugar" name="submit" >
<input type="submit" value="Reiniciar" name="submit" >
<input type="submit" value="Volver" name="submit" >
</form>
```

El código de jugar php se podría (debería) hacer algo de este tipo

```
//Evaluamos las posibles acciones según la opción que me lleva a este fichero
switch ($_POST['submit']) {
```

```

    case 'Reiniciar': //He presionado reset de jugar.php
        //Acciones
    case 'Empezar': //Vengo del indx
        //Acciones
        break;
    case 'Jugar': //He presionado jugar
        //Acciones
        break;
    case 'Volver': //He presionado volver, quiero ir al index
        //Acciones
        break;
    default: //He accedido directamente a este fichero sin pasar por index ?
        Quiero permitirlo ???
}

```

Esta es una buena práctica de programación que permite hacer un código legible y controlado

Buenas prácticas 2

En la medida de lo posible debemos usar los valores de los inputs para evitar código

En el ejemplo que nos ocupa tenemos tres opciones que normalmente habéis puesto en los radio.

En el index

```

<form action="jugar.php" method="POST">
    <input type="radio" name="num_intentos" value=10 <br />
    <input type="radio" name="num_intentos" value=15 <br />
    <input type="radio" name="num_intentos" value=20 <br />
    <input type="submit" value="Empezar" name="submit">
</form>

```

Si cada radio tiene como valor este valor, en jugar.php, para todos los casos, tanto al empezar, como reiniciar los valores serán

```

$num_intentos = filter_input(INPUT_POST, "num_intentos");
$min = 1;
$max = pow(2, $num_intentos);
$num = intdiv($max, 2);

```

Esto indistintamente de la opción seleccionada, ahora mucho código y aporta legibilidad

Buenas prácticas 3

Siempre siempre separamos el código (controlador) de la vista.

Mirar <https://es.wikieducator.org/Usuario:ManuelRomero/ProgramacionWeb/Sintaxis/Conceptos>



Empecemos aplicar el modelo vista **controlador**

- Podemos llevar este criterio desde el principio siguiendo el esquema siguiente

Esqueleto MVC

```
<!-- Empecemos nuestro fichero escribiendo instrucciones php  
Estas instrucciones realizarán cálculos y obtendrán valores  
En ningún momento generan salidas,  
Trataremos de almacenar los valores en variables  
-->  
<?php  
//instrucciones php  
$variables = "Mensaje escrito desde php";  
?>  
<!--Ahora el código html-->  
<!DOCTYPE html>  
<html>  
  <head>  
    <meta charset="UTF-8">  
    <title>probando php</title>  
  </head>  
  <body>  
    <!--Dentro del html cuando necesitemos ver valores  
    los escribimos  
    También puede ser que necesitemos instrucciones  
    las escribimos, pero intentando escribir lo necesario.  
    Esto es vista  
    -->  
    <h1> <?php echo $variable ?></h1>  
  </body>  
</html>  
?>
```

Esto lo penalizaré bastante, pues me parece fundamental para que nuestro código sea profesional. **NO SE TRATA SÓLO DE HACER UN CÓDIGO QUE FUNCIONE**, hay que **hacerlo bien, legible y mantenible**.

Buenas prácticas 4

Al hilo del apunte anterior, nunca se debe de generar texto antes del body. Nosotras programamos en back, pero generamos una página html. Ésta ha de pasar validadores de html con 100% de corrección, si no, google, **nunca** indexará nuestras páginas.

Todos los *echo* o *print*, etc (funciones o primitivas que escriban texto en el fichero) **siempre** dentro del <body>

Buenas prácticas 5

El código hay que comentarlo, pero bien comentado. Por ejemplo:

```
//La variable $edad contendrá la edad que introduzca  
$edad = 5;
```

Esto no es un comentario válido, no aporta nada

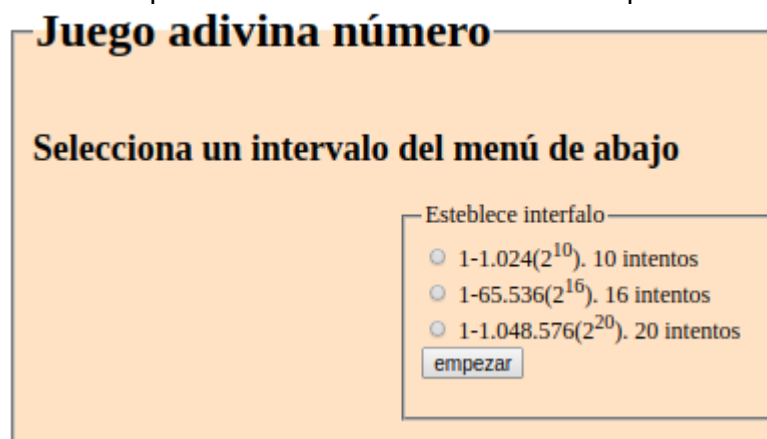
Los comentarios han de ser significativos apropiados y un resumen de una parte del código que ayuden a entenderlo

```
//Tenemos códigos de error identificados en el fichero error.php
//Al final se muestran, solo para desarrollo, quitar en producción.
$error = 5;
```

Buenas prácticas 6

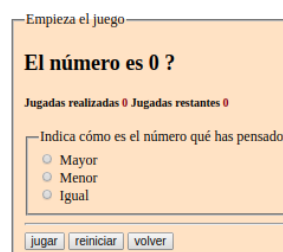
La aplicación no debe de fallar nunca, si el usuario ha de elegir una opción para que la aplicación funcione y no la selecciona el programa no puede ser que se quede colgado, debe de estar preparado para estas situaciones.

No sólo debemos programar lo que tiene que hacer cuando todo va bien, sino estar preparado para el resto de opciones que se pueden producir aunque no sean la dinámica normal. En el ejemplo que nos ocupa si tenemos el index que el usuario ha de seleccionar una opción



Qué pasa si el usuario da a **empezar** y no ha seleccionado nada

No debería de ocurrir esto, ya que no es un uso de nuestra aplicación



Ni mucho menos esto

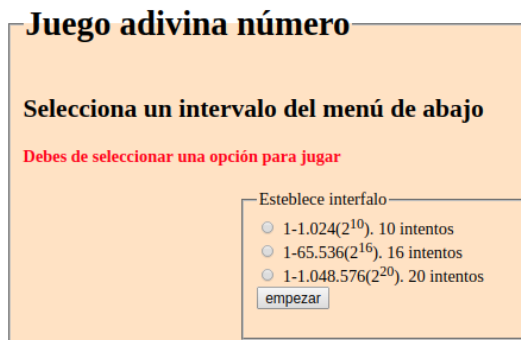


Esta página no funciona

La página **localhost** no puede procesar esta solicitud ahora.

HTTP ERROR 500

Por ejemplo, en este caso, podría mostrar esto:



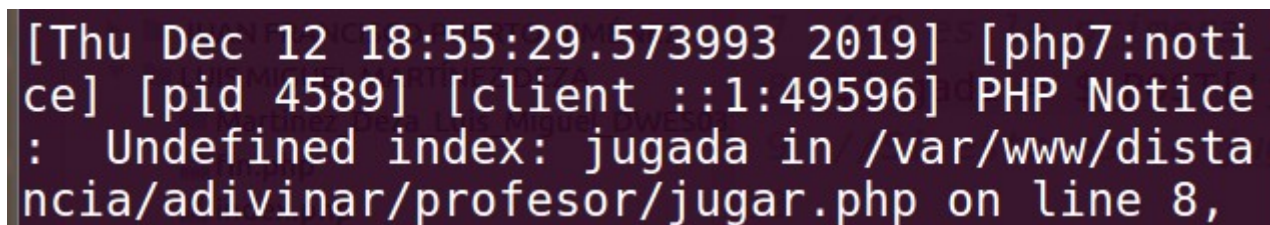
Buenas prácticas 7

Cuando asigno un valor a una variable, lógicamente éste ha de existir , si no habrá un warning y se asignará el valor null.

Por ejemplo si yo hago

```
$jugada = $_POST['jugada']
```

Y resulta que no tengo valor de `$_POST` ya que vengo del index, saltará un warning que puedo ver en error.log



Me dice que el índice `jugada` no está definido, ya que vengo de `index.php` y no de `jugar.php`

Supongamos que si no existe el índice de `$_POST['jugada']` queremos dar a la variable `$jugada` el valor 0, Para evitar el warning (notice) , tendríamos que comprobar que existe el valor de ese índice antes de asignarlo, algo del tipo

```
if (isset($_POST['jugada']))
    $jugada = $_POST['jugada'];
else
    $jugada=0;
```

O bien en una línea con una expresión ternaria

```
$jugada =(isset($_POST['jugada']))? $_POST['jugada']: 0;
```

O bien, mucho mejor, usar la nueva funcionalidad de **php 7** llamado *operador de fusión null*

```
$jugada = $_POST['jugada'] ?? 0;
```

Su lectura es que si la expresión primera, en este caso `$_POST['jugada']` es null devuelve el segundo valor `0`, si no devuelve el valor de la primera expresión.

Buenas Prácticas 8

Normalmente cuando hacemos un `header()` que es una función que escribe cabeceras http, y por lo tanto va a entregar un nuevo recurso al cliente, suele ocurrir que ya no tiene sentido que se siga

ejecutando el script actual, por lo que es profesional hacer un **exit** o bien un **die** (se consideran alias) para abortar el script y la generación de página actual que no se va a utilizar para nada.

Estos constructores del lenguaje puede recibir un operador que sería el texto que mostrarán en su ejecución. En este caso no lo veré ya que hay un header delante.

Podría ocurrir que quiera hacer cosas después del header como anotar datos en una base de datos o en un fichero, en cuyo caso no he de hacer el exist para que eso se realice. En este caso por ejemplo si ya he superado el número de intentos quiero terminar

```
if ($jugada > $num_intentos) {//Si ya llevamos mas jugadas de las permitidas terminamos  
    header("Location: fin.php?jugada=$jugada");  
    exit;  
}
```

o bien

```
if ($jugada > $num_intentos) {//Si ya llevamos mas jugadas de las permitidas terminamos  
    header("Location: fin.php?jugada=$jugada");  
    die;  
}
```