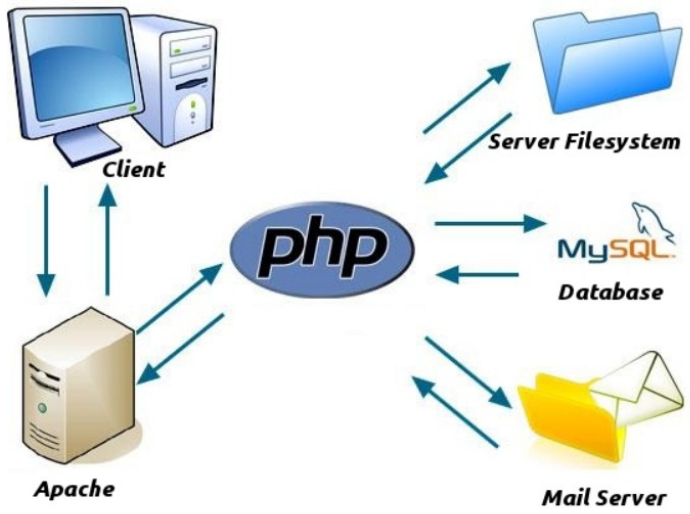
U.D.2.- Características da linguaxe PHP

Sumario

Introdución

PHP, orixinalmente o acrónimo de *"Personal Home Pages"* e posteriormente o acrónimo recursivo de "*PHP Hypertext Preprocesor*" é unha linguaxe de programación de servidor: é executado no servidor e o resultado (normalmente en formato HTML) é recibido polo navegador do cliente:



Foi creado no 1994 como un subconxunto de scripts en Perl creados por Rasmus Lerdof. Posteriormente foi modificado por outros programadores ata a versión actual, que é a 8.0 (neste 2021).

Para poder traballar con PHP temos que ter instalado un servidor de http/https co Apache. Pode ser Apache, Nginx, etc.

Na aula traballaremos en linux con Docker e contenedores, mentres que en Windows unha opción rápida pode ser empregar XAMPP.

Hai moitos sitios web nos que podes ter acceso a información de php, pero o máis usado polos programadores, e onde podemos obter toda a información que empregaremos no presente curso, pode ser o sitio oficial:

https://www.php.net/manual/es/

ou algo máis resumido:

https://manuais.iessanclemente.net/index.php/PHP

Inserción de código

Fíxate que o código PHP insírese dentro do código HTML, empregando as etiquetas de apertura e peche **<?php** e **?>**, respectivamente.

```
<!DOCTYPE html>
<html>
<head>
        <meta charset="UTF-8" />
        <title>Primeiro exemplo</title>
</head>
<body>
      <h2> Primeiro exemplo en PHP </h2>
      <?php
            echo "Ola mundo<br>";
      ?>
</body>
</html>
```

Sentencias

- As sentencias teñen que rematar con ‘;’
- A última sentencia do bloque non precisa do ‘;’ pero é recomendable.
- Agruparemos varias sentencias coas chaves { .... }, por exemplo dentro dun bucle **for**

Variables

As variables comezan co símbolo de $, e non é preciso definilas antes de usalas, como nalgunhas linguaxes de programación.

Ollo coas minúsculas e maiúsculas, é distinto $apelido que $Apelido.

O nome da variable só pode comezar por unha letra ou por barra baixa: ‘_’

O resto dos caracteres pode ser calquera letra, números ou ‘_’

```
<!DOCTYPE html>
<html>
<head>
        <meta charset="UTF-8" />
        <title>Variables</title>
</head>
<body>
      <h2> Variables </h2>
      <?php
            $a=5;
            $_b=7.35;
            $animal33=”Pantera Rosa”;

            echo $animal33, “<br>”, $a, “<br>”, $_b;
      ?>
</body>
</html>
´
```

Cada variable é visible no ámbito no que está definida, teremos variables locais dentro de cada función, e variables globais co seu ámbito fóra das funcións. Para que as variables globais se poidan empregar dentro da función empregaremos a palabra reservada **global.** Veremos exemplos disto polo miúdo cando vexamos as funcións.

Variables predefinidas

Existen algunhas variables predefinidas , gardando información do entorno de execución do intérprete e do propio PHP.
As máis empregadas, todos array asociativos, son:
$_GET[‘variable’]: variables recibidas por GET
$_POST[‘variable’]: variables recibidas por POST
$_SERVER[‘variable’]: todas as variables de alcance global
$_COOKIE[‘variable’]: variables pasadas a través de cookies
$_FILES[‘variable’]: ficheiros pasados a través de POST
As principais variables do servidor están nun array asociativo **$_SERVER:** *$_SERVER[‘SERVER_NAME’], _SERVER[‘SERVER_PORT’], $_SERVER[‘SERVER_SOFTWARE’], $_SERVER[‘SERVER_REMOTE_PORT], $_SERVER[‘REMOTE_ADDR’]*, etc.

Por exemplo, se temos un formulario que envía por GET, con 2 input de tipo texto, con name igual a “nome” e “apelido”, un programa que captura os valores introducidos e os amosa por pantalla podería ser:

```
<!DOCTYPE html>
<html>
<head>
        <meta charset="UTF-8" />
        <title>Paso por GET</title>
</head>
<body>
      <h2> Variables </h2>
      <?php
            $onome=$_GET[‘nome’]; //GARDAMOS O nome NOUTRA VARIABLE

            echo $onome <br>, $_GET[‘apelido’]; /* E O apelido DIRECTAMENTE */
      ?>
</body>
</html>
```

Tipos de datos en PHP

Xa vimos que as variables en PHP comezan sempre polo símbolo $. O seu tipo é asignado automaticamente, pero pode cambiar se cambia o seu contido.
Os tipos de datos simples en PHP son:

- ✔ **booleano** (*boolean*). Os seus posibles valores son **true** y **false** . Ademais, calquera número enteiro considérase como *true* , salvo o 0 que é *false* .
- ✔ **entero** (*integer*). Calquera número sen decimais. Pódense representar en formato decimal, octal (comenzando por un 0), ou hexadecimal (comenzando por 0x).
- ✔ **real** (*float*). Calquera número con decimais. Pódense representar tamén en notación científica.
- ✔ **cadena** (*string*). Conxuntos de caracteres delimitados por comiñas simples o dobres.
- ✔ **null**. Es un tipo de datos especial, que se usa para indicar que la variable non ten valor.

Por exemplo:

```
$oBooleano = false;
$oEnteiro= 0x2A;
$oReal = 2.35;
$aCadea=”ola que tal?”;
$a = null;
```

Cando se fan operacións con variables de distintos tipos, ambas convértense primeiro a un tipo común. Por exemplo, se sumamos un enteiro a un real, o enteiro convértese primeiro a real antes de facer a suma:

```
$enteiro=4;
$real= 3.7;
$resultado = $enteiro + $real;  // O resultado será real, e valerá 7.7
```

Tamén podemos realizar a conversión de xeito forzado:

```
$enteiro=4;
$real= 3.7;
$resultado = $enteiro + (int) $real;  // Agora a variable $real convértese a enteiro
                        //antes de sumarse. Así, $ resultado será enteiro e valerá 7
```

Existen funcións predefinidas para comprobar o tipo de dato. Todas teñen unha sintaxe parecida:

- ***is_bool()*** : Comproba se unha variable é de tipo booleano .
- ***is_float()*** : Comproba se unha variable é de tipo es float .
- ***is_numeric()*** : Comproba se unha variable é de tipo número ou un string que se pode converter a número.
- ***is_string()*** : Comproba se una variable é de tipo string .
- ***is_array() :*** Comproba se una variable é un array .
- ***is_object()*** : Comproba se una variable é un obxecto.

Todas devolven *true* ou *false*. Un exemplo sería:

```
$a = 5;
if (is_int($a))
      echo “ A variable é enteira”; // Mostrarase a mensaxe
```

Operadores

Os operadores aritméticos serán os mesmos que en Javascript: + , - , * , / , %

Ademais do operador de asignación = están permitidos os operadores de asignación += , -= , *= , /=.

Os operadores de comparación son ==, != , < , > , <=, >=. Ademais, existe o operador === que compara o valor e tipo. E !== . Así:

```
12 === 12.0  // é falso porque non coincide o tipo de dato
10 === 10  //será verdadeiro
```

O operador de concatenación para as cadeas é o operador punto (.)

Por exemplo:

```
$cad1 = "Ola,  ";
$cad2 = "que tal";
$cadea = $cad1 . $cad2;   //$cadea valerá 'Ola, que tal'
```

Estruturas de control

Alternativa simple: if, else

De forma similar a Javascript:

```
<!DOCTYPE html>
<html>
<head>
        <meta charset="UTF-8" />
        <title>Alternativa if-else</title>
</head>
<body>
      <h2> Variables </h2>
      <?php
            $num = $_GET['numero1'];
            if ($num < 5)
                  echo "O número é menor que 5";
            else
                  echo "Número maior ou igual que 5";
      ?>
```

```
</body>
</html>
```

Alternativa múltiple: if, elseif

A diferencia con outras linguaxes é que o elseif non ten espazo polo medio (~~else if~~):

```
<!DOCTYPE html>
<html>
<head>
        <meta charset="UTF-8" />
        <title>Alternativa if-else</title>
</head>
<body>
      <h2> Variables </h2>
      <?php
            $num=$_GET['numero1'];
            if ($num < 5)
                  echo "O número é menor que 5";
            elseif ($num == 5)
                  echo "O número é igual a 5";
            else
                  echo "Número maior que 5";

      ?>
</body>
</html>
```

Alternativa con switch

Cando a comparación é sempre coa mesma variable e con ==, podemos empregar a sentencia switch:

```
<!DOCTYPE html>
<html>
<head>
        <meta charset="UTF-8" />
        <title>Alternativa switch</title>
</head>
```

```
<body>
      <h2> Variables </h2>
      <?php
            $num=$_GET[‘numero1’];
            switch ($num)
            {
            case 1: echo “Número igual a 1”;
                    break;
            case 3: echo “Número igual a 3”;
                    break;
            case 5: echo “Número igual a 5”;
                    break;
            default: echo “Número distinto a 1,3 e 5”;
            }
      ?>
</body>
</html>
```

Bucle 1. while

O bucle while repítese mentres a condición sexa certa, Sintaxe similar a Javascript ou C:

```
<!DOCTYPE html>
<html>
<head>
        <meta charset="UTF-8" />
        <title>While. Conta atrás </title>
</head>
<body>
      <h2> Variables </h2>
      <?php
            $num=$_GET[‘numero1’];
            while ($num > 0)
            {
                  echo $num, “<br>”;
                  $num--;  //dentro do bucle DEBE cambiar a variable da condición
            }
            echo “Chegamos ao 0!”;
      ?>
</body>
</html>
```

Bucle 2. do_while

Agora a condición avalíase ao final, polo que o bucle sempre se executa unha vez:

```
<?php
      $num=$_GET[‘numero1’];
      do
      {
            echo $num, “<br>”;
            $num--;      //dentro do bucle cambia a variable da condición
      } while ($num > 0);     //Obrigatorio o ‘;’
      echo “Chegamos ao 0!”;  ?>
```

Bucle 3. for

```
<?php
            for ($i=0; $i < 20; $i++)
                  echo $i,”<br>”;
?>
```

Bucle4. foreach

Veremos o **foreach** na sección dos **arrays**.

Arrays

Os arrays serán un tipo de dato que asocian claves e valores. Serán principalmente de 2 tipos:

INDEXADOS: a clave é numérica, un ÍNDICE numérico. Empezamos a contar no 0 :

```
$notas = [ 8.5, 6, 9 ];
echo  $notas[0];  // Mostrará o 8.5
echo $notas[1] ; // Mostrará o 6
echo $notas[2 ; // Mostrará o 9
```

8.5	6	9
[0]	[1]	[2]

ASOCIATIVOS: a clave é unha cadea de texto.

```
$arrayAsociativo = [
      'nome' => 'Federico',
      'apelidos' => 'Caeiro',
      'idade' => 17 ];
echo $arrayAsociativo['nome'] // Mostrará Federico
echo $arrayAsociativo['apelidos'] // Mostrará Caeiro
echo $arrayAsociativo['idade'] // Mostrará 17
```

'Federico'	'Caeiro'	17
['nome']	['apelidos']	['idade']

Como crear un array

Existen varios xeitos de crear un array en PHP

1.- Con **corchetes**:

Podemos crear un array como fixemos arriba:

```
$notas = [ 8.5, 6, 9 ];   //array Indexado

$arrayAsociativo = [                //array asociativo
```

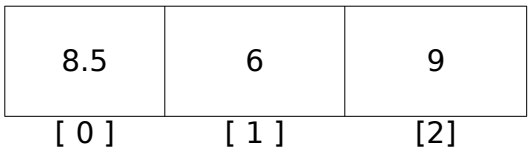
```
            'nome' => 'Federico',
            'apelidos' => 'Caeiro',
            'idade' => 17 ];
```

2.- Co construtor ***array:***

```
      $notas = array( 8.5, 6, 9 );   //array Indexado

$arrayAsociativo = array(                //array asociativo
            'nome' => 'Federico',
            'apelidos' => 'Caeiro',
            'idade' => 17 );
```

3.- Elemento a elemento:

```
      $notas[0]=8.5;
      $notas[1]=6;
      $notas[2]=9;

      Ou ben:

      $notas[0]=8.5;
      $notas[]=6;
      $notas[]=9;
```



Por defecto, nos arrays o primeiro elemento é o 0. Canto utilizamos a notación $notas[] o índice será o seguinte ao maior índice numérico xa asignado. Así, se creamos un array así:

```
      $cidade[3]="París";
      $cidade[]="Londres;
      echo $cidade[4]; // Amosará Londres
```

E mesmo se están combinados:

```
$datos = array(3 => 17, 27, 55, 'c' => 121, 8);
```

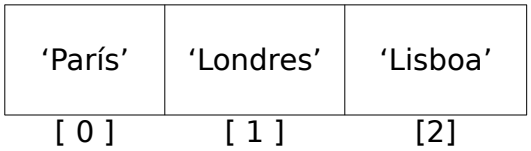
```
echo $datos[3]."<br>"; // mostra 17
echo $datos[4]."<br>"; // Amosará 27
echo $datos[5]."<br>";// Amosará 55
echo $datos[6]."<br>";// . OLLO: Amosará 8
echo $datos['c']."<br>"; // Amosará 121
```

Mostrar contido dun array

Normalmente percorreremos o **array** cos bucles **for** e **foreach**.

Nos array indexados é máis claro empregar o for:

```
//ARRAY INDEXADO
$cidade=array("París", "Londres", "Lisboa");
for ($i=0; $i<3; $i++)
      echo $cidades[$i]."<br>";
```



Aínda que menos empregado, tamén se pode empregar o foreach:

```
      //ARRAY INDEXADO
      $cidade=array("París", "Londres", "Lisboa");
      foreach($cidade as $valor)
            echo $valor;
```

Nos arrays asociativos é máis claro empregar o foreach:

```
      //ARRAY ASOCIATIVO
      $capital=array("Francia"=>"París",
                  "Inglaterra"=>"Londres",
                  "Portugal"=>"Lisboa");

      foreach($capital as $pais=>$cidade)
            echo "A capital de $pais é $cidade";
```

Se queremos imprimir unha posición dada, podemos empregar as chaves: {}

```
      //ARRAY ASOCIATIVO
      $capital=array("Francia"=>"París",
                  "Inglaterra"=>"Londres",
```

```
                "Portugal"=>"Lisboa");
echo "A capital de Francia é {$capital["Francia"]}"; //ou empregar o '.' e concatenar:
echo " A capital de Francia é ". $capital["Francia"];
```

Para amosar o contido do array completo amosando o contido podemos empregar as funcións

```
print_r($capital);  //os contidos
var_dump($capital);  //tamén o tipo dos datos que contén cada posición
```

Eliminar elemento dun array: unset( ) e array_values( )

Para eliminar un elemento dun array, empregaremos *unset()*, e logo empregaremos *array_values()* para reordenar o array (eliminamos o elemento borrado e reordenamos os índices):

```
//ARRAY INDEXADO
$cidade=array("París", "Londres", "Lisboa");

//ELIMINAMOS O SEGUNDO ELEMENTO
unset($cidade[1]);
$cidade = array_values($cidade);
```

Funcións

As funcións son bloques de código que poden ser reutilizados. A ese bloque darémoslle un nome para poder chamar á función desde diferente sitios.
Empregamos a sentencia ***function*** seguida do nome da función. Os parámetros que recibe chámanse argumentos, e o bloque de sentencias irá entre chaves:

```
function nomeFuncion($parametro1, $parametro2, ...){
      ..................;
      ..................;
return $variable;
}
```

Se a función retorna un valor ao sitio desde que é chamada levará a sentencia **return $variable;**

Funcións definidas polo usuario

As funcións definidas polo usuario deberán estar no corpo do documento, ou ben agrupadas nunha librería (onde se definen varias funcións nun mesmo ficheiro). Vexamos unha función ***suma*** que calcula a suma de 2 números e devolve a súa suma.

```
function suma ($num1, $num2) {
$resultado= $num1+ $num2;
return $resultado;
}
```

A función recibe os argumentos ou parámetros (neste exemplo os 2 números: $num1, $num2) e devolverá este resultado. Podemos chamar a función suma desde o noso código php:

```
<?php
function suma($num1, $num2){  //DEFINIMOS A FUNCIÓN
            //DENTRO DA FUNCIÓN AS VARIABLES A USAR SON $num1 e $num2
            $resultado= $num1+ $num2;
            return $resultado;
}
            $res=suma(4,5);  //CHAMAMOS Á FUNCIÓN ENVIANDO OS VALORES 4 E 5
            echo “O resultado de sumar 4 e 5 é “,$res;
```

```
            $n1=$_GET[“numero1”];
            $n2=$_GET[“numero2”];
            $res=suma($n1,$n2); //CHAMAMOS Á FUNCIÓN ENVIANDO VALORES DE $n1 e $n2
            echo “O resultado de sumar $n1 e $n2 é $res”;
?>
```

Podemos definir varias funcións, que podemos chamar dentro do noso código:

```
<?php
      function suma($num1, $num2){
                  //DENTRO DA FUNCIÓN AS VARIABLES A USAR SON $num1 e $num2
                  $resultado= $num1+ $num2;
                  return $resultado;
      }

      function media($numero1, $numero2, $numero3){
                  $resultadoMedia=($numero1+$numero2+$numero3)/3;
                  return $resultadoMedia;
      }

      $n1=$_GET[“numero1”];
      $n2=$_GET[“numero2”];
      $n3=$_GET[“numero3”];
      $res=suma($n1,$n2); //CHAMAMOS Á FUNCIÓN ENVIANDO VALORES DE $n1 e $n2
      echo “O resultado de sumar $n1 e $n2 é $res”;

      $res=media($n1,$n2,$n3); //ENVIAMOS AGORA OS 3 VALORES
            echo “A media de $n1, $n2 e $n3 é $res”;

?>
```

Procedementos

Se a función non devolve nada (non ten a sentencia ***return***) é considerado un **procedemento**. Empregaranse para agrupar sentencias que aparecen varias veces no noso código, e por simplicidade do código. Por exemplo:

```
<?php
      function saudamos($nomeCompleto)
      {
      echo “Benvido, $nomeCompleto”;
      }

      $usuario=$_GET[“usuario”];
```

```
	saudamos($usuario];
?>
```

Librerías de funcións

Podemos agrupar funcións nun ficheiro con extensión .php, no que se coñece como unha **librería**. Para poder utilizaras librerías podemos usar as palabras reservadas ***include*** ou ***require*****,** ou ***require_once***.

Por exemplo, podemos gardar as funcións anteriores nun único ficheiro **libreriaSumaMedia.php:**

```
<?php
	function suma($num1, $num2){
	//DENTRO DA FUNCIÓN AS VARIABLES A USAR SON $num1 e $num2
		$resultado= $num1+ $num2;
		return $resultado;
	}

	function media($numero1, $numero2, $numero3){
		$resultadoMedia=($numero1+$numero2+$numero3)/3;
		return $resultadoMedia;
	}
	?>
```

Agora poderíamos utilizar esas funcións noutro ficheiro php:

```
<?php
	include("libreriaSumaMedia.php");
	$n1=$_GET["num1"];
	$n2=$_GET["num2"];
	$n3=$_GET["num3"];

	echo "A suma dos dous números vale ", suma($n1,$n2);
	echo "A media dos dous números vale ", media($n1,$n2);
?>
```

Cadeas de texto

As cadeas de texto son datos do tipo "string", formados por caracteres. En ***php*** as cadeas poden estar delimitadas por comiñas simples ou por comiñas dobres.

Comiñas dobres

Todas as variables que están dentro se expandirán. Para mostrar caracteres especiais (\n, \t, \$,...) empregamos a barra invertida. Tamén para unha comiña ou o nome dunha variable:

```
<?php
            $idade=22;
            echo "Ten $idade anos";

            // Mostrará por pantalla: Ten 22 anos

            echo "A variable  \$idade: $idade";
            //Sairá por pantalla: A variable  $idade: 22
?>
```

Tamén se pode rodear entre chaves:

```
<?php
            $ano = 60;
            echo “Viviu nos {$ano}s";
            //Sairá por pantalla:  Viviu nos 60s.

?>
```

Comiñas simples

Dentro das comiñas simples non se expanden as variables

```
<?php
      $num = 25;
      echo'Num: $num';
      //a salida é: Num: $num
?>
```

Só se pode mostrar como caracteres especiais a comiña e a barra \, es:

```
      <?php
```

```
        $num = 25;
        echo'Num: \\ $num  \'';
        //a salida é: Num: \ $num'  A variable non se expande
?>
```

Funcións para o manexo de cadeas

- **echo (arg1 [, arg2...]):** non é realmente unha función (é unha sentencia da linguaxe, non precisa paréntese).
- **int strlen (string cadea):** devolve o número de caracteres da cadea.
- **string substr (string cadea, int comezo [, int lonxitude]) :** Devolve unha subcadea, empezando polo comezo e de lonxitude lonxitude.
- **string strstr(string cadea, string busca) :** devolve a cadea desde a primeira aparición da cadea busca
- **string strchr(string cadea, string letra) :** idéntica á anterior, pero a primeira aparición da letra
- **string strrchr(string cadea, string letra):** devolve a cadea desde a última aparición do carácter
- **int strpos(string cadea, string busca):** devolve a posición numérica da primeira aparición.
- **string str_replace(string buscada, string substituída, string orixinal):** substitúe as aparicións da cadea buscada na cadea orixinal pola substituída.
- **string substr_replace(string cadea, string substituida, int comezo):** substitúe a cadea pola cadea
- **string trim (string cadea):** elimina os espazos á esquerda e dereita da cadea.
- **string ltrim (string cadea):** elimina os espazos á esquerda da cadea.
- **string rtrim (string cadea):** elimina os espazos á dereita da cadea.
- **string strtoupper(string cadea):** pasa a maiúsculas todos os caracteres dun texto
- **string strtolower(string cadea):** pasa a minúsculas todos os caracteres dun texto

- **array explode(string separador, string cadea [, int límite]):** devolve un array que contén en cada posición do mesmo as partes da cadea separadas polo separador.
- **string ucfirst (string cadea):** a maiúsculas o primeiro carácter da cadea
- **string ucwords (string cadea):** a maiúsculas cada palabra.
- **int strcmp(string $str1, string $str2):** devolve un enteiro. Devolve < 0 se str1 é vai antes alfabeticamente que str2; >0 se str2 vai antes alfabeticamente que str1, 0 se son iguais.
- **string urlencode(string $str):** devolve unha cadea codificada para pasar variables a unha páxina php
- **string urldecode(string $str):** decodifica calquera cifrado %## dunha cadea dada (suponse que foi previamente codificada para ser pasada a outra páxina php).
- Etc.

Podes ver a sintaxe de máis funcións en **https://www.php.net/manual/es/ref.strings.php**

Funcións de cadeas de carácter multibyte

Hai que considerar que as cadeas con acentos gárdanse en cadeas multibyte (varios bytes por carácter). Como en castelán e en galego hai moitas palabras con acentos é conveniente empregar estas funcións. Son similares ás anteriores empezando polo prefixo **mb_:**

mb_strlen( ), mb_strstr( ), mb_strpos(), etc.

Podes ver a sintaxe en: **https://www.php.net/manual/es/ref.mbstring.php**

PHP 8. Cadeas de caracteres

A **versión 8** engade algunhas funcións para buscar cadeas de caracteres:

- **bool str_contains(string cadea, string busca) :** devolve *true* se o string busca está no string cadea.

Ex: str_contains(“Ola mundo!”,”mundo”); //Devolverá *true*

- **bool str_starts_with(string cadea, string busca)**: devolve *true* se o string *cadea* empeza pola cadea *busca*

- **bool str_ends_with(string cadea, string busca)**: devolve *true* se o string *cadea* remata pola cadea *busca*