El llenguatge PHP (I): Paradigma procedural

Sintaxi bàsica

Un script PHP normalment té l'extensió ".php" i, en el cas de fer-lo servir en un entorn web, conté etiquetes HTML mesclades amb codi PHP pròpiament dit, el qual ha d'estar escrit entre l'etiqueta d'inici "**<?php**" i l'etiqueta de final "**?>**". Aquest codi PHP no arriba mai al navegador: és processat prèviament per l'intèrpret PHP (executat mijtjançant un servidor web, normalment remot), el qual retorna el resultat d'aquest processament en forma de codi HTML suplementari llest per ser "digerit", llavors sí, pel navegador. El següent codi mostra un exemple mínim (on el codi HTML es mostra en color negre i el codi PHP en color verd)...:

```
<!DOCTYPE html>
<html><body>
<h1>Un exemple bàsic</h1>
<?php
echo("Hola");
?>
</body></html>
```

...i el següent codi representa el que rebrà el navegador, resultat d'haver-se processat el codi anterior:

```
<!DOCTYPE html>
<html><body>
<h1>Un exemple bàsic</h1>
Hola
</body></html>
```

Elements que cal tenir en compte a l'hora d'escriure codi PHP són:

*Totes les ordres PHP han de finalitzar amb un punt i coma ("**;**")

*Hi ha diferents sintaxis possibles per incloure comentaris en el codi: els comentaris d'una sola línia poden començar amb "//" o bé "#" però els comentaris de més d'una línia han de començar amb "**/***" i acabar amb "***/**", com es pot veure en els següents exemples:

```
//Això és un comentari d'una sola línia
#Això també, independent de l'anterior
/*Això és un comentari
   de més d'una línia */
```

NOTA: Un comentari és un tros de text escrit entre codi PHP que no és llegit/executat per l'intèrpret. El seu propòsit és que sigui vist per la persona que estigui llegint el codi per tal d'explicar i afegir context al codi que acompanya. També es poden fer servir per deshabilitar temporalment parts de codi per a no tenir-les en compte.

*Els noms de les paraules reservades del llenguatge PHP (*if*, *else*, *while*, *echo*,...), els de les classes i funcions predefinides del llenguatge així com els de les classes i funcions definides per l'usuari són "case-insensitive"; és a dir, tant és que s'escriguin en majúscula com en minúscula. En canvi, els noms de les variables són "case-sensitive"; és a dir, si per exemple tenim *$var, $Var , $VAR*,... cadascuna d'elles seran tractades com a variables diferents independents entre sí.

NOTA: Tot el que s'explica en aquest document (i moltíssim més!) està perfectament documentat de forma oficial a https://www.php.net/manual/es/langref.php

Variables

Una variable és un "calaix" que permet emmagatzemar informació. Per treballar amb elles a un codi PHP cal indicar sempre el seu nom (inventat per nosaltres de manera que sigui descriptiu de la raó de ser de la variable) precedit sempre del símbol dólar ("**$**"). Declarar una variable (és a dir, crear-la, reservant-se l'espai adient a la memòria de l'ordinador) i inicialitzar-la (és a dir, assignar-li un determinat valor) són dos processos que a PHP es realitzen sempre a la vegada, tal com es mostra al següent codi d'exemple:

```
<?php
$unaVar="Hola";     //Aquesta variable emmagatzema un valor de cadena (ha d'anar escrita entre cometes dobles)
$unaAltra=4;        //Aquesta variable emmagatzema un valor sencer
$unaMes=5.67;       //Aquesta variable emmagatzema un valor decimal
$iUnaAltra=true;    //Aquesta variable emmagatzema un valor booleà
echo("Els valors de les variables són $unaVar $unaAltra $unaMes $iUnaAltra");
$unaVar="Adeu";   //Un cop creada, es pot modificar el seu valor en qualsevol moment (d'aquí el nom de "variable")
echo("El valor de 'unaVar' ara és $unaVar");
?>
```

NOTA: Els noms de les variables no poden començar per un número i només poden contenir els caràcters A-z, 0-9 i _ . D'altra banda, són case-sensitive!

Per a què la funció *echo()* de PHP afegeixi un salt de línia (normalment al final del text imprimit), es pot indicar la seqüència "\n", així: *echo("Hola\n");*. No obstant, si el text està pensat per veure's inserit dins de codi HTML, s'ha d'optar per una alternativa, com seria fent servir l'etiqueta d'aquest llenguatge dissenyada per això, que és "*
*", així: *echo("Hola
");*.

Una altra seqüència interessant és "\t" (per incloure un tabulador). D'altra banda, si es volen escriure dins de la cadena els símbols ", ', \ i $ tal quals, cal precedir-los d'una contrabarra per a què perdin el seu significat especial, així: \" , \' , \\ i \$ (és el que se'n diu "escapar" un caràcter).

Tot i que sovint no cal, es pot usar el punt per concatenar cadenes, així: *echo("Hola" . $unaVar . "Adéu");* Diem que no cal perquè si les variables escrites entre cometes dobles, l'intèrpret PHP sempre les interpretarà (una altra cosa seria si estiguessin escrites entre cometes simples; en aquest cas l'intèrpret PHP no interpreta res i escriu els caràcters que formen el seu nom tal qual són).

Existeix a PHP una altra funció molt similar a *echo* anomenada *print.* La major diferència és que la primera no té cap valor de retorn i la segona sí.

Cal tenir en compte que PHP és un llenguatge parcialment dèbilment tipat. Això vol dir que una mateixa variable pot allotjar en un moment donat un valor d'un tipus i en un altre moment un valor diferent d'un altre tipus. No obstant, ho és parcialment perquè en el cas dels paràmetres de funcions (i els seus valors de retorn) sí que es pot (i, de fet, és recomanable) indicar el seu tipus. En qualsevol cas, els diferents tipus de dades que PHP sap reconèixer només són els següents: cadenes (*string*), números sencers (entre -2147483648 i 2147483647 ; *int*), números decimals (*float*), valors booleans (*bool*), vectors (*array*), el tipus *null* i objectes (els quals estudiarem més endavant).

NOTA: Els números sencers es poden expressar també en notació hexadecimal, així: *$num=0x4F3A;*. D'altra banda, els números decimals es poden expressar també en notació científica, així: *$num=-4.35e-6*;
NOTA: Les variables que tenen assignat el valor *null* són de tipus *null.* Això a la pràctica implica que aquestes variables, en realitat, no tenen cap valor assignat. Es pot "buidar" una variable simplement assignant-li aquest valor: *$unaVar=null;*

La funció *var_dump()* serveix per veure el tipus i valor d'una determinada variable prèviament inicialitzada (incloent arrays, que els estudiarem més endavant), així: *var_dump($unaVar);* D'altra banda, la funció *print_r($unaVar)* és similar però la seva sortida és més fàcilment llegible pels humans.

Altres funcions que ens seran força útils amb el treball amb variables són *isset($unaVar);* i *unset($unaVar);* La primera retorna el valor booleà *true* si la variable passada com a paràmetre ja està definida (per comprovar el contrari, si la variable no està definida, es pot usar l'operador de negació "**!**", com per exemple així: *if (!isset($unaVar))*. La segona "desdefineix" (és a dir, destrueix de la memòria) la variable indicada, de manera que una invocació posterior a *isset()* retornarà el valor booleà *false*.

Arrays

Un array és una variable que pot emmagatzemar múltiples valors a la vegada. En el següent exemple, fem servir la funció *array()* per crear un array (que anomenem *$cars*) amb tres valors inicials:

```
<?php
$cars = array("Volvo","BMW","Toyota");
var_dump($cars);
?>
```

Emmagatzemar els valors anteriors en sengles variables "simples" es podria haver fet així...

```
$cars1 = "Volvo";
$cars2 = "BMW";
$cars3 = "Toyota";
```

...però llavors, a banda de fer servir moltes més variables (¿què passaria si tinguéssim 34567 cotxes en lloc de 3?), no hi ha una manera de recórrer la llista de cotxer per trobar-ne un d'específic. La gràcia dels arrays és que sota el mateix i únic nom (el de l'array) cada valor individual està referenciat per un índex, índex que ens permetrà recórrer tots els elements de l'array d'una forma molt senzilla, com de seguida veurem.

Depenent del tipus d'índex, podem dir que en PHP hi ha tres tipus d'arrays:

*Arrays indexats: els seus índexs són números sencers començant pel 0 (0,1,2...)
*Arrays associatius: els seus índexs són claus (és a dir, els elements són parelles clau<->valor)
*Arrays multidimensionals: contenen un o més arrays al seu interior (també coneguts com "matrius")

L'array creat a l'exemple anterior és de tipus indexat. Hi ha dues maneres de crear aquest tipus d'arrays: o bé amb la funció *array()* tal com hem vist o bé assignant "manualment" els valors desitjats a cada element individual, així (com es pot veure, l'índex s'indica entre claudàtors):

```
<?php
$cars[0] = "Volvo";
$cars[1] = "BMW";
$cars[2] = "Toyota";
var_dump($cars);
?>
```

En qualsevol cas, sigui com s'hagi creat l'array, un cop el tinguem definit amb diferents valors, per tal d'accedir a un d'ells en concret només caldrà indicar el seu índex corresponent, tal com es pot veure en el següent exemple:

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
```

NOTA: Si volguéssim incloure un element d'un array directament dins d'una cadena (i no com es fa a l'exemple anterior concatenant-lo mitjançant l'operador "."), en lloc d'escriure, com inicialment podríem pensar, *echo "I like $cars[0], $cars[1] and $cars[2] ";*, a versions antigues de l'intérpret PHP calia escriure *echo "I like {$cars[0]}, {$cars[1]} and {$cars[2]} ";* La raó d'indicar les claus era per delimitar tot el que té a veure amb la definició de l'element en qüestió, ja que sense elles l'intèrpret PHP considerava que la variable -inexistent- *$cars* anava seguida de cadenes com "[0]","[1]",etc sense trobar relació entre una cosa i l'altra.

Si parlem dels arrays associatius, també tenim dues maneres de crear-los; així...:

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
var_dump($cars);
?>
```

...o així:

```
<?php
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
var_dump($age);
?>
```

En qualsevol cas, sigui com s'hagi creat l'array associatiu, un cop el tinguem definit amb diferents valors, per tal d'accedir a un d'ells en concret només caldrà indicar la seva clau corresponent, tal com es pot veure en el següent exemple:

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo "Peter is " . $age['Peter'] . " years old.";
?>
```

Per recórrer tots els elements d'un array haurem de fer servir algun bucle com *for* o *foreach*, que els estudiarem més endavant. D'altra banda, convé saber que existeix la funció count($nomArray); , la qual retorna la longitud (és a dir, el nombre d'elements) de l'array indicat (ja sigui indexat o associatiu), com es pot veure en aquest exemple:

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo count($cars);
?>
```

> NOTA: Als exemples anteriors no s'ha vist, però a PHP és perfectament possible que cada element d'un mateix array sigui d'un "tipus" diferent (una cadena, un sencer, etc) degut precissament a què PHP és un llenguatge dèbilment tipat
> NOTA: Per consultar una referència completa de totes les funcions relacionades amb arrays, es pot visitar https://www.w3schools.com/php/php_ref_array.asp ; aquesta web conté una breu descripció i exemples d'ús de cada funció. D'altra banda, per conéixer específicament funcions dissenyades per ordenar array, es pot visitar https://www.w3schools.com/php/php_arrays_sort.asp i per estudiar com gestionar arrays multidimensionals, es pot visitar https://www.w3schools.com/php/php_arrays_multidimensional.asp

Constants

Una constant és, a l'igual que una variable, un "calaix" que permet emmagatzemar informació però, a diferència de les variables, una vegada que s'ha definit un valor en una constant, aquest valor ja no podrà canviar (ni "desdefinir-se")

> NOTA: Els noms de les constants no poden començar per un número ni pel símbol $, i només poden contenir els caràcters A-z, 0-9 i _ . D'altra banda, són case-sensitive! (a no ser que s'indiqui el contrari amb la funció *define()*, veieu més avall)
> NOTA**:** A diferència de les variables, les constants són automàticament globals a tot l'script. Això vol dir que es pot utilitzar una constant dins d'una funció, fins i tot si es defineix fora de la funció. Ho estudiarem quan veiem les funcions més endavant

Per crear una constant, s'utilitza la funció *define("nomConstant", valor);* El valor assignat es pot especificar literalment o pot ser el valor actual d'alguna variable o el valor de retorn d'una funció. Per defecte, els noms de les constants distingeixen entre majúscules i minúscules, però això pot canviar si afegim un tercer paràmetre a la funció *define()* amb el valor booleà *true*. També es pot crear una constant de tipus array utilitzant aquesta funció i fent servir la notació especial de claudàtors:

```
<?php
define("cars", ["Alfa Romeo", "BMW", "Toyota" ]);
echo cars[0];
?>
```

És interessant saber l'existència de la funció *defined("nomConstant");* , la qual és l'equivalent a la funció *isset()* però per a constants, ja que retorna *true* si la constant indicada està definida (i *false* si no).

El llenguatge PHP incorpora un conjunt de constants predefinides. Totes estan llistades a https://www.php.net/manual/en/reserved.constants.php però les més rellevants són:

__DIR__ : La ruta de la carpeta on el fitxer interpretat està ubicat. Cal tenir en compte que aquest fitxer és el que conté la línia de codi actual (per tant, si és un fitxer inclós via *include()* -o similar- informarà del nom d'aquest fitxer inclós)
__FILE__ : El nom del fitxer que s'està interpretant. Cal tenir en compte el mateix
__LINE__ : Número de línia de codi actualment en execució. Cal tenir en compte el mateix
PHP_EOL : Caràcter de nova línia independent del sistema operatiu i, per tant, vàlid per tots ells (això és degut a que cada sistema operatiu en fa servir un de diferent -en Linux per exemple és "\n"-)
PHP_OS : Nom del sistema operatiu on l'script s'està interpretant
PHP_VERSION : Versió de l'intèrpret PHP utilitzat
M_PI : Constant numèrica equivalent al número PI. Totes les constants matemàtiques tenen un nom que comença per M_*

Operadors

Els operadors s'utilitzen per realitzar operacions sobre variables i valors. Podem classificar-los així:

Aritmètics (+, -, *, /, %, **)
D'assignació (=, +=, -=, *=, /=, %=)
De comparació (==, ===, *!=*, *!==*, >, <, >=, <=, <=>)
D'increment/decrement (++, --)
Lògics (*and* i &&, *or* i ||, *not* i *!*)
De cadena (. , .=)
D'array (+, ==, ===, *!=*, *!==*)
D'assignació condicional (*?: , ??*)
De tipus bitwise (&, |, ^, ~, <<, >>)

Per conéixer alguns exemples pràctics sobre el seu ús, es pot consultar https://www.w3schools.com/php/php_operators.asp o http://www.hackingwithphp.com/3/12/3/complete-operator-list . Si l'expressió en qüestió conté diversos operadors, la seva precedència segueix les normes indicades aquí: http://www.hackingwithphp.com/3/12/7/operator-precedence-and-associativity D'altra banda, també convé tenir en compte les taules mostrades a https://www.php.net/manual/en/types.comparisons.php, les quals informen del resultat de comparacions segons els tipus de les dades involucrades.

Condicionals: *if/else*

Sovint voldreu que un codi realitzi diferents accions segons es compleixi una determinada condició o una altra, o una altra...o fins i tot cap. Per escriure declaracions condicionals, en general s'utilitzarà la sentència *if*, la qual executa el bloc de codi que hi ha escrit al seu interior només si la condició que tingui associada és certa. La seva sintaxi general és aquesta:

if (condició) { codi a executar si la condició és certa; }

El següent exemple mostrarà la frase "Hola!" si l'hora actual és menor de 20:

```
<?php
$t = date("H");
if ($t < 20) {
    echo "Hola!";
}
?>
```

La sentència *if...else* executa un bloc de codi si la condició associada és certa i un altre bloc de codi diferent si és falsa. La seva sintaxi general és aquesta:

```
if (condició) {  codi a executar si la condició és certa; }
else          {  codi a executar si la condició és falsa; }
```

El següent exemple mostrarà la frase "Hola!" si l'hora actual és menor de 20 i "Adéu!" si no:

```
<?php
$t = date("H");
if ($t < 20) {
    echo "Hola!";
} else {
    echo "Adéu!";
}
?>
```

La sentència *if...elseif...else* és capaç d'avaluar en cadena múltiples condicions que estan escrites en ordre i executar el bloc de codi associat a la primera d'aquestes condicions que sigui certa. És a dir, si la primera condició és certa, executarà el seu bloc de codi associat i ja està però si no, avaluarà la segona condició: si aquesta és certa executarà el seu bloc de codi associat i ja està però si no, avaluarà la tercera condició: si aquesta és certa...etc. La seva sintaxi general és aquesta:

```
if (condició)      { codi a executar si la condició és certa; }
elseif (una altra condició) { codi a executar si la condició anterior és falsa i aquesta és certa; }
...
else               { codi a executar si totes les condicions anteriors eren falses;  }
```

El següent exemple mostrarà la frase "Hola!" si l'hora actual és menor de 10, "Què tal?" si és menor de 20 (però no era menor de 10) i "Adéu!" si no és ni una cosa ni l'altra (és a dir, si és igual o major de 20).

```
<?php
$t = date("H");
if ($t < 10) {
   echo "Hola!";
} elseif ($t < 20) {
   echo "Què tal?";
} else {
   echo "Adéu!";
}
?>
```

Condicionals: *switch*

La sentència *switch* és utilitzada per indicar estructures de tipus *if...elseif...else* però permet escriure-les d'una forma més directa i senzilla. La seva sintaxi general és aquesta:

```
switch ($nomVariable) {
    case valor1:
        codi a executar si $nomVariable=valor1;
        break;
    case valor2:
        codi a executar si $nomVariable=valor2;
        break;
     ...
    default:
      codi a executar si el valor de $nomVariable és diferent a tots els anteriors;
}
```

Així és com funciona: primer tenim una única expressió (la majoria de vegades, però, serà simplement el nom d'una variable), que s'avalua una vegada. A continuació, es compara el valor de l'expressió amb els valors de cada "cas" de l'estructura. Si hi ha una coincidència, s'executa el bloc de codi associat amb aquest "cas". Cal utilitzar la directiva *break* per evitar que el codi continuï executant-se automàticament pels casos següents. La sentència (opcional) *default* s'utilitza si no es troba cap coincidència.

```
<?php
$favcolor = "red";
switch ($favcolor) {
    case "red":
        echo "Your favorite color is red!";    break;
    case "blue":
        echo "Your favorite color is blue!";  break;
    default:
        echo "Your favorite color is neither red, blue, nor green!";
}
?>
```

Bucles: *while*

Sovint voldreu que un mateix codi s'executi una i altra vegada. En lloc d'escriure el mateix bloc de codi repetit un darrera l'altre, podem utilitzar unes estructures anomenades bucles. Concretament, el bucle *while*, per exemple, executarà un bloc de codi sempre que la condició especificada sigui certa.

while (condició és certa) { codi a executar; }

L'exemple següent primer assigna el valor 1 a la variable $x ($x=1). Tot seguit, s'avalua la condició indicada al bucle *while*, el qual executarà el bloc de codi que conté sempre que $x sigui menor o igual que 5. Per tant, només començar aquesta condició és certa, així que aquest bloc de codi s'executa, bloc que, a més de mostrar un missatge per pantalla, fa augmentar en 1 el valor de $x ($x++). Un cop finalitzat el bloc, es torna "a dalt" per tornar a avaluar la condició de nou. Si continua essent certa (de moment sí, perquè ara $x=2), es tornarà a executar el bloc, amb la qual cosa el valor de $x augmentarà de nou en 1 (en aquest cas fins arribar a valer 3) i es tornarà "a dalt" a avaluar de nou la condició, etc. Aquesta roda acabarà quan $x=6.

```
<?php
$x = 1;
while($x <= 5) {
   echo "The number is: $x <br>";
   $x++;
}
?>
```

El bucle *do...while* és similar a l'anterior, només que sempre executarà el bloc de codi com a mínim un cop tot i que la condició inicialment sigui falsa, perquè s'avalua al final (i no al principi com amb *while*)

do { codi a executar; } while (condició és certa);

L'exemple següent primer assigna el valor 1 a la variable $x. Tot seguit, mostra un missatge, fa augmentar en 1 el valor de $x i finalment s'avalua la condició indicada. Si aquesta és certa, es torna a "dalt" del bloc i es repeteix el missatge i es torna a augmentar en 1 el valor de $x per finalment tornar a avaluar la condició. I així fins que aquesta sigui falsa (si ho acaba essent, que en aquest cas sí).

```
<?php
$x = 1;
do {
   echo "The number is: $x <br>";
   $x++;
} while ($x <= 5);
?>
```

Bucles: *for*

El bucle *for* s'utilitza quan se sap ja d'entrada quants cops es vol que s'executi el bloc a repetir. És una manera més còmoda d'escriure un cas concret de bucle *while*. La seva sintaxi general és aquesta, on...:

for ($counter=nº; condició $counter; increment $counter) { codi a executar; }

$counter=nº : Estableix el valor inicial (nº) d'una variable que farà de comptador
condició $counter : Condició que s'evalua a cada iteració del bucle: si és certa, el bloc de codi s'executarà (i en acabar tot seguit es tornarà a avaluar la condició de nou per repetir el cicle); si no, el bucle acaba. Un exemple podria ser *$counter < 10*
increment $counter : Modifica el valor de la variable comptador (ja sigui augmentant-lo en un o més, o disminuïnt-lo en un o més). Aquest pas es fa just abans d'avaluar la condició a cada iteració (excepte la priemra vegada, que és quan s'inicialitza el comptador)

L'exemple següent mostra els números del 0 al 10 (recordem que l'operador "++" fa augmentar el valor de la variable afectada en 1 cada cop que s'interpreta):

```
<?php
for ($x = 0; $x <= 10; $x++) {  echo "The number is: $x <br>";  }
?>
```

Sabent abans la seva longitud, amb un bucle *for* també es poden recórrer els elements d'un array:

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
$arrlength = count($cars);
for($x = 0; $x < $arrlength; $x++) {
   echo $cars[$x];
   echo "<br>";
}
?>
```

Bucles: *foreach*

El bucle *foreach* només serveix per una cosa: per recórrer arrays sense haver de saber prèviament la seva longitud. La seva sintaxi és:

foreach ($array as $valor) { codi a executar; }

Per a cada iteració de bucle, el valor de l'element actual s'assignarà a la variable *$valor* i el punter es mourà llavors a l'element següent (fins que arribi a l'últim element de l'array). L'exemple següent mostra els valors de l'array donat (en aquest cas, *$colors*):

```
<?php
$colors = array("red", "green", "blue", "yellow");
foreach ($colors as $valor) {
   echo "$valor <br>";
}
?>
```

També es pot utilitzar en arrays associatius, així:

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
foreach($age as $clau => $valor) {
   echo "Key=" . $clau . ", Value=" . $valor;
   echo "<br>";
}
?>
```

Tant dins el bucle *for* com dins el bucle *foreach* es pot utilitzar la sentència *break;* la qual en provoca la sortida i, per tant, deixant sense executar les eventuals repeticions (iteracions) que encara quedarien per finalitzar el bucle. També existeix la sentència *continue;* la qual provoca la sortida de la repetició actual, continuant inmediatament per la següent.

Funcions definides per l'usuari

Una funció és un bloc de codi que té assignat un nom, de forma que pugui ser utilitzat a partir de llavors repetidament en un programa només invocant a aquest nom. Per tant, tenim d'una banda la definició de la funció (a l'inici del programa) i de l'altra les seves possibles múltiples "crides" (o execucions) al llarg del programa. La definició de la funció es realitza amb aquest sintaxis:

function nomFunció() { codi a executar; }

NOTA: Els noms de les funcions han de començar per una lletra o un _ (no pas un número) i són case-insensitive. La idea, d'altra banda, és que el seu nom reflecteixi de forma senzilla la seva funcionalitat

En l'exemple següent creem una funció anomenada *writeMsg()* definint-la amb el codi indicat entre les claus i, tot seguit, l'executem (com es pot veure, la crida consisteix simplement en escriure el seu nom seguit de parèntesis):

```
<?php
//Definició de la funció
function writeMsg() {
    echo "Hello world!";
}
//Crida a la funció
writeMsg();
?>
```

Es poden enviar dades a funcions en el moment de la seva crida mitjançant l'ús de paràmetres. Un paràmetre és similar a una variable, només que s'indiquen entre els parèntesis que apareixen darrera el nom de la funció (separats entre sí per comes): en el cas de la definició s'indicarà el nom del paràmetre i en el cas de les crides s'indicarà el valor concret que es vol que tingui aquell paràmetre per la crida en qüestió. Per exemple, el següent codi té una funció anomenada *familyName()* amb un paràmetre anomenat *$fname*; cada cop que aquesta funció és cridada, se li passa un valor concret ("Jani", "Hege", etc), que serà assignat com a valor del paràmetre *$fname* per tal de procedir a continuació a l'execució del codi intern de la funció.

```
<?php
function familyName($fname) {
    echo "$fname Refsnes.<br>";
}
familyName("Jani");
familyName("Hege");
familyName("Stale");
?>
```

El següent exemple utilitza una funció amb dos paràmetres (*$fname* i *$year*):

```
<?php
function familyName($fname, $year) {
    echo "$fname Refsnes. Born in $year <br>";
}
familyName("Jani", 1983);
familyName("Hege", 1975);
familyName("Stale", 1978);
?>
```

Noteu com a l'exemple anterior no s'ha indicat el tipus de dada que cada paràmetre contindrà. Això pot provocar múltiples errors d'interpretació (per exemple, si es passa un valor decimal a una funció que requereix un nombre sencer, aquest valor es truncarà a sencer automàticament, o si es passa una cadena que comença per dígits també es truncarà a sencer automàticament descartant la resta de caràcters de la cadena que no siguin dígits, etc). Per evitar això, es pot fer que l'intèrpret PHP sigui estricte en el tipus de dada dels paràmetres simplement indicant el tipus desitjat a la seva declaració (els principals tipus són *int*, *float*, *string*, *bool*, *array* i objecte, tot i que a https://www.php.net/manual/es/language.types.intro.php es pot veure que hi ha algun més), així:

*function nomFunció(**tipus** $nomParam1, **tipus** $nomParam2,...) { codi a executar; }*

NOTA: Cal saber que a versions antigues de l'intèrpret PHP, la declaració anterior només funcionava si la primera línia de tots els fitxers on es cridava a la/es funció/ns en qüestió (és a dir, just després de l'etiqueta *<?php*, era una línia tal com ***declare(strict_types=1);***

Fent aquesta declaració estricta de tipus, si es produís una discrepància de tipus, es dispararà un "Error fatal" i sabrem que alguna cosa no funciona com es desitja; això ajuda a no causar problemes de diagnòstic aleatoris i confusos. En el següent exemple, s'intenta sumar un número decimal amb una cadena que comença per un dígit...sense declaració estricta el resultat hagués sigut "12.8" però amb ella salta un error.

```
<?php
function addNumbers(float $a, int $b) {
   $c= $a + $b;
   echo "El resultat és $c";
}
echo addNumbers(7.8, "5 days");
?>
```

D'altra banda, també és possible implementar paràmetres opcionals els quals, si no s'indiquen, agafen el valor per defecte que s'hagi indicat en la declaració de la funció. Per exemple, el següent codi mostra com si cridem a la funció setHeight() sense indicar cap valor, el paràmetre $minheight valdrà automàticament 50:

```
<?php
function setHeight(int $minheight = 50) {
    echo "The height is : $minheight <br>";
}
setHeight(350);
setHeight(); // will use the default value of 50
setHeight(135);
setHeight(80);
?>
```

Les funcions no només poden admetre paràmetres sinó que també poden retornar un valor en finalitzar l'execució del seu codi, valor que pot ser directament utilitzat a l'script principal just en el lloc on s'ha fet la crida a la funció o bé ser recollit, per exemple, per alguna variable per així guardar aquest resultat. Per fer-ho cal utilitzar la sentència *return*, així:

```
<?php
function sum(int $x, int $y) {
   $z = $x + $y;
   return $z;
}
echo "5 + 10 = " . sum(5, 10) . "<br>";
echo "7 + 13 = " . sum(7, 13) . "<br>";
echo "2 + 4 = " . sum(2, 4);
?>
```

Igual que passava amb els paràmetres, amb el valor de retorn també es pot forçar el seu tipus; fent això, ens assegurem que només valors del tipus esperat són retornats. La sintaxis necessària per això és la següent:

function nomFunció(tipus $nomParam1, tipus $nomParam2,...) ***: tipus*** *{ codi a executar; }*

Un exemple el tenim en el següent codi:

```
<?php
function addNumbers(float $a, float $b) : string {
   $c= $a + $b;
   return "El resultat és $c";
}
echo addNumbers(1.2, 5.2);
?>
```

NOTA: Els tipus indicats als paràmetres i als valors de retorn poden anar precedits del símbol "**?**" (és a dir, *?int* en lloc de *int*, *?float* en lloc de *float*, etc.). Això significa que, a més del tipus especificat, NULL també es pot passar com a argument (o retornar com a valor, respectivament). És el que se'n diu tipus "nullables"

Àmbit de les variables

En PHP, les variables poden ser declarades en qualsevol lloc de l'script però segon on hagin sigut declarades no podran ser utilitzades a tot arreu. "L'àmbit" de la variable és la zona de l'script on una variable concreta podrà ser referenciada (és a dir, usada). A PHP hi ha tres àmbits possibles:

*Local: Quan una variable és declarada dins d'una funció. Només pot ser accedida dins de la funció
*Global: Quan una variable és declarada fora d'una funció. Només pot ser accedida fora de la funció
*Estàtic: Variable local que no s'elimina en acabar l'execució de la funció sinó que es manté (mantenint el valor per properes execucions de la mateixa funció)

Veiem un exemple de variable local:

```
<?php
function myTest() {
  $x = 5; // local scope
  echo "<p>Variable x inside function is: $x</p>";
}
myTest();
// using x outside the function will generate an error
echo "<p>Variable x outside function is: $x</p>";
?>
```

NOTA: Es poden tenir variables locals amb el mateix nom en diferents funcions, ja que aquestes variables només són reconegudes per la función on són declarades

I un exemple de variable global:

```
<?php
$x = 5; // global scope
function myTest() {
   // using x inside this function will generate an error
   echo "<p>Variable x inside function is: $x</p>";
}
myTest();
echo "<p>Variable x outside function is: $x</p>";
?>
```

NOTA: Es poden tenir variables locals amb el mateix nom que variables globals perquè no se solapen. Això es pot veure per exemple executant el codi següent, en es mostra la cadena "baz" perquè la única variable de la qual és conscient la instrucció *echo* és la global.

```php
<?php
   function foo() { $bar = "wombat";  }
   $bar = "baz";
   foo();
   echo("$bar");
?>
```

Si es vol accedir a una variable global des de dins d'una funció, es pot indicar al principi del codi de la funció en qüestió amb quines de les variables globals concretes s'hi vol treballar mitjançant la directiva *global*, així:

```php
<?php
$x = 5;
$y = 10;
function myTest() {
  global $x, $y;
  $y = $x + $y;
}
myTest();
echo $y;  // outputs 15
?>
```

D'altra banda, cal saber que PHP emmagatzema totes les variables globals que haguem definit en el nostre script dins d'un array predefinit anomenat *$GLOBALS["nomVariable"]* La gràcia és que aquest array també és accessible des de l'interior de les funcions i, per tant, pot ser utilitzat, per exemple, per actualitzar el contingut d'aquest tipus de variables directament des dins de la funció, o per llegir el valor d'una variable global per assignar-lo a una variable local, etc. A continuació es mostra un exemple similar a l'anterior però fent ús d'aquesta funcionalitat:

```php
<?php
$x = 5;
$y = 10;
function myTest() {
   $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
}
myTest();
echo $y; // outputs 15
?>
```

NOTA: L'array *$GLOBALS* és només un dels arrays que PHP té definits de forma predeterminada i que són accessibles arreu dels nostres scripts, incloent l'interior de funcions i objectes. Aquests arrays, anomenats conjuntament "superglobals" són concretament, a més del ja conegut *$GLOBALS*, els següents:

$_GET	Contains all variables sent via a HTTP GET request. That is, sent by way of the URL.
$_POST	Contains all variables sent via a HTTP POST request.
$_FILES	Contains all variables sent via a HTTP POST file upload.
$_COOKIE	Contains all variables sent via HTTP cookies.
$_REQUEST	Contains all variables sent via HTTP GET, HTTP POST, and HTTP cookies. This is basically the equivalent of combining $_GET, $_POST, and $_COOKIE, and is less dangerous than using $GLOBALS. However, as it does contain all variables from untrusted sources (that is, your visitors), you should still try to steer clear unless you have very good reason to use it.
$_SESSION	Contains all variables stored in a user's session.
$_SERVER	Contains all variables set by the web server you are using, or other sources that directly relate to the execution of your script.
$_ENV	Contains all environment variables set by your system or shell for the script.

NOTA: En el cas del superglobal $_SERVER, teniu una descripció detallada de les elements que pot incloure a https://www.w3schools.com/php/php_superglobals_server.asp

I un exemple de variable estàtica on es pot veure que cada cop que es crida la funció, la variable -que és local, recordem-ho!- encara manté el valor de la darrera crida:

```
<?php
function myTest() {
  static $x = 0;
  echo $x;
  $x++;
}
myTest();
myTest();
myTest();
?>
```

NOTA: A destacar el concepte d'assignació de valors a variabless per "referència", explicat aquí: http://www.hackingwithphp.com/3/10/0/references Ho estudiarem en profunditat més endavant, però.

Includes i requires

Les sentències *include("/ruta/fitxer");* o *require("/ruta/fitxer");* "copien-peguen" allà on són invocades tot el contingut textual/de codi/de marques que hi hagi dins del fitxer referenciat. La idea és que contingut comú (text,HTML/CSS o PHP) que calgui repetir en diversos scripts (per exemple, encapçalaments, peus, menús...) es pot deixar en un fitxer a incloure posteriorment allà on sigui necessari. Ambdues sentències són idèntiques; només les diferencia la seva reacció davant fallades (com per exemple que no es trobi el fitxer a incloure, o que no es tinguin permisos per llegir-lo, etc): fent servir la primera només es produeix un "warning" i l'script-base continua executant-se mentre que fent servir la segona es produeix un error E_COMPILE_ERROR si s'atura l'execució de l'script-base.

A continuació es presenta un exemple, on d'una banda tenim l'arxiu "llibreria.php" amb el següent contingut...

```
<?php
function per2($a) {
  $b=$a*2;
  return($b);
}
echo("Llibreria inclosa");
?>
```

...i de l'altra un script PHP qualsevol que fa ús del codi existent dins de "llibreria.php" (la qual, en no indicar cap ruta absoluta concreta, hauria de trobar-se a la mateixa carpeta que el propi script PHP: la ruta relativa indicada sempre serà respecte la de l'script que vol incloure el contingut extern, la qual, com ja sabem, la podem obtenir fàcilment mitjançant la variable predefinida *__DIR__*):

```
<?php
require("llibreria.php");
$valor=per2(34);
echo($valor);
?>
```

També existeixen les sentències *require_once("/ruta/fitxer");*/*include_once("/ruta/fitxer");*, que són respectivament similars a *require()*/*include()* només amb la diferència que si el fitxer indicat ja ha sigut inclós prèviament, no es tornarà a fer

NOTA: Teniu més exemples d'aquestes sentències a https://www.w3schools.com/php/php_includes.asp

Gestió d' errors

Un aspecte que hem de tenir molt en compte que ens posem a escriure scripts PHP és com controlar i gestionar els possibles errors que poguem cometre a l'hora de desenvolupar codi. Afortunadament, l'intèrpret PHP proporciona diverses eines que ens poden ajudar a detectar, identificar (i, per tant, resoldre) els errors apareguts.

Per començar, hem de saber que el comportament general de l'intèrpret PHP davant l'ocurrència d'un error es pot configurar a l'arxiu "php.ini" corresponent (depenent de si és de tipus FPM, o de tipus CLI, etc). Concretament, algunes de les directives més rellevants en aquest àmbit són les següents (teniu la llista completa a https://www.php.net/manual/en/errorfunc.configuration.php) :

NOTA: Recordeu que per conéixer el valor actual de les directives de configuració de l'intèrpret PHP es pot visualitzar el resultat de la funció predeterminada *phpinfo();* en una pàgina web o bé executar la comanda *php -i* en un terminal. En ambdós casos, per cada directiva apareixeran dos valors, un al costat de l'altre: l'anomenat "local" i l'anomenat "master". El darrer és el valor establert a l'arxiu "php.ini" i el primer és el valor que actualment està en efecte, el qual pot ser igual que el "master" o bé un altre si aquest ha sigut sobreescrit mitjançant la funció *ini_set("nomdirectiva","valor")*, la qual serveix precisament per modificar "al vol" per l'script on s'executi determinades directives de configuració

* *error_reporting* : Indica el nivell de gravetat a partir del qual les notícies, advertències i errors apareixeran (ja sigui a la pantalla o en els logs). El seu valor, tot i ser numèric, se sol indicar amb una o més constants mnemotècniques combinades. Algunes d'aquestes combinacions són les següents (teniu la llista completa a https://www.php.net/errorfunc.constants) :

  E_ERROR (*=1*): S'informarà de tots els errors fatals (és a dir, l'script s'aturarà) ocorreguts en temps d'execució
  E_WARNING (*=2*): S'informarà de totes les advertències (és a dir, l'script no s'aturarà) ocorreguts en temps d'execució
  E_PARSE (*=4*): S'informarà dels errors d'interpretació de codi (previs a la seva execució)
  E_DEPRECATED (*=8192*): S'informarà, en temps d'execució, de les notícies relacionades amb la troballa de codi que en futures versions de l'intèrpret deixarà de funcionar
  E_ALL (*=32767* o *-1*) : S'informarà de totes les notícies, advertències i errors.

  Els valors anteriors es poden combinar amb els operadors "bitwise" (recordem que "&" equival a "AND", "|" equival a "OR", "~" equival a "NOT"...per més informació consulteu https://www.php.net/manual/en/language.operators.bitwise.php). Per exemple, un valor com "*E_ALL & ~E_DEPRECATED & ~E_PARSE"* voldria dir que es vol informar de tots els errors (i warnings, etc) excepte els de tipus "deprecated" i els d'interpretació.

* *display_errors* : Indica si els errors (a partir del nivell indicat a la directiva *error_reporting*) dels nostres scripts PHP es mostraran en el navegador web. El seu valor pot ser *On* o *Off* Aquesta directiva és interessant activar-la (per defecte està desactivada) quan estem desenvolupant algun projecte web però convindrà desactivar-la en posar-lo en producció per tal de donar més informació del compte als posibles visitants sobre les interioritats del nostre codi

* *display_startup_errors* : Indica si els errors (a partir del nivell indicat a la directiva *error_reporting*) a l'hora de llegir l'arxiu "php.ini" per part de l'intèrpret PHP en posar-se en marxa es mostraran en el navegador web. El seu valor pot ser *On* o *Off* (per defecte és *Off*)

* *log_errors* : Indica si els errors (a partir del nivell indicat a la directiva *error_reporting*) dels nostres scripts PHP es guardaran en un fitxer de log (on l'usuari "www-data/apache" hi tingui permisos d'escriptura i la ruta del qual s'indicarà amb la directiva *error_log*) o bé directament en el Journal del sistema (si *error_log* té el valor especial "syslog"). El seu valor pot ser *On* o *Off*

  NOTA: Si la directiva *log_errors* val *On* però la directiva *error_log* no té cap valor assignat, els missatges d'error s'enviaran a "stderr" s'usa l'intèrpret PHP de consola o l'arxiu "error.log" genèric de l'Apache si s'usa l'intèrpret PHP-FPM

NOTA: A Fedora, per defecte el valor de *log_errors* a "/etc/php.ini" val *On* i el valor de *error_log* (establert a l'arxiu "/etc/php-fpm.d/www.conf" mitjançant la sintaxi *php_admin_value[nomDirectiva]*) és "**/var/log/php-fpm/www-error.log**" A Ubuntu per defecte el valor de *log_errors*, tant a a "/etc/php/*X.Y*/fpm/php.ini" com a "/etc/php/*X.Y*/cli/php.ini", també val *On* però el valor de *error_log*, no està establert (a l'arxiu "/etc/php/*X.Y*/fpm/pool.d/www.conf" apareix la línia *php_admin_value[error_log]=**/var/log/fpm-php.www.log*** però està comentada)

D'altra banda, el llenguatge PHP disposa de les següents funcions predefinides que permeten, per exemple, controlar l'execució del nostre codi en el moment que es detecti un error i/o registrar-ne explícitament la seva ocurrència (en un fitxer de log, o remotament a una màquina, o mitjançant l'enviament d'un correu, etc per fer-ne un estudi posterior), entre altres tasques. Algunes d'aquestes funcions són les següents (teniu la llista completa a https://www.php.net/manual/en/ref.errorfunc.php) :

* *die("missatge")* o *exit("missatge")* : Finalitza l'execució de l'script PHP inmediatament, mostrant, a més (i tal com faria la comanda *echo*) el missatge indicat a pantalla. Molt utilitzada en comprobacions on, en detectar algun determinat valor no desitjat en alguna condició, es vol interrompre l'script.

* *set_error_handler("nomFunció")* : A partir de què aparegui aquesta funció en un codi, cada cop que es detecti un error s'executarà automàticament la funció que s'hagi indicat com a paràmetre (funció que haurà d'estar definida prèviament per nosaltres i haurà de tenir obligatòriament quatre paràmetres, per ordre: el número d'error ocorregut, el missatge d'error corresponent, la ruta del fitxer on aquest error ha ocorregut i la línia de codi on ha ocorregut)

* *trigger_error("missatge")* : Genera "artificialment" un error (amb la qual cosa, es pot desembocar, si així està definida, en l'execució d'una eventual funció disparada mitjançant *set_error_handler()*). Aquesta funció és util quan s'indica en codi executat sota determinades circumstàncies. El missatge indicat és el que apareixerà a la sortida. Opcionalment es pot afegir un segon paràmetre que pot valer E_USER_WARNING o E_USER_ERROR (per defecte val E_USER_NOTICE).

* *error_log("missatge")* : Envia el missatge indicat al destí que estigui especificat a la directiva *error_log* (que pot ser un determinat fitxer de log o el Journal del sistema) A Fedora, per exemple, serà l'arxiu **/var/log/php-fpm/www-error.log"**, tal com s'ha comentat a la nota anterior. Molt utilitzada per detectar determinats estats (no desitjats) del nostre programa durant l'execució del codi

---

En el cas que estiguem desenvolupant un codi orientat a objectes (tal com veurem pròximament), el mecanisme de control d'errors més complet, segur i flexible (i, per tant, més habitual) és, no obstant, la gestió de les anomenades "excepcions". Les estudiarem quan aprenem com programar amb objectes.

Funcions pròpies del llenguatge PHP

El llenguatge PHP disposa "de fàbrica" de multitud de funcions "ja preparades" que serveixen per una gran varietat de tasques diferents. La llista completa (amb tota la documentació associada: funcionalitat, paràmetres admesos, valors de retorn, casos d'ús, etc) es troba a https://www.php.net/manual/en/funcref.php i caldria tot un curs per coneixer-les i dominar-les, però en podem destacar breument les següents:

* Relacionades amb la gestió de cadenes (https://www.php.net/manual/en/book.strings.php): Aquí ens podem trobar funcions com *strlen("cadena");*, que retorna el número de caràcters de la cadena indicada, o *str_word_count("cadena");* que retorna el número de paraules, o *strrev("cadena");*, que retorna la cadena indicada al revés, o *strpos("cadena1","cadena2", n);*, que retorna la posició del primer caràcter de "cadena2" -recordem que les posicions comencen per 0- que es trobi dins de "cadena1" (fent la comparació de forma case-sensitive) a partir de la posició indicada en el tercer paràmetre (o des del principi de "cadena1" si aquest no s'indica) o *false* si no es troba "cadena2" dins de "cadena1" (la variant case-insensitive d'aquesta funció és *stripos("cadena1","cadena2");*), o *str_replace("cadena1","cadena2","cadena3",n);*, que reemplaça totes (o només la quantitat que

s'hagi indicat com a valor numèric del seu opcional quart paràmetre) les ocurrències (detectades de forma case-sensitive) de "cadena1" per "cadena2" dins de "cadena3" (la variant case-insensitive és *str_**i**replace("cadena1","cadena2","cadena3",n);*), o *substr("cadena", x, y);*, que retorna una subcadena de la cadena indicada com a primer paràmetre que comenci a partir de la posició -començant per 0- indicada com a segon paràmetre i que tingui una quantitat de caràcters indicada com a tercer paràmetre (si aquest tercer paràmetre no s'indica, s'entendrà que la longitud serà fins el final de la cadena), o *ltrim("cadena");*, *rtrim("cadena");*, *trim("cadena");* , les quals eliminen els espais en blanc, tabuladors i salts de línia inicials, finals o existents a tots dos extrems, respectivament, de la cadena indicada, o *str_pad("cadena",n,"x", CONSTANT);*, que omple els caràcters que li falten a la cadena indicada al primer paràmetre per arribar a la longitud indicada al segon paràmetre amb el caràcter indicat al tercer paràmetre en el costat indicat al quart paràmetre (*STR_PAD_LEFT,STR_PAD_RIGHT,STR_PAD_BOTH*), o *strtolower("cadena");*, *strtoupper("cadena");*, que converteixen en minúscules o majúscules, respectivament, la cadena indicada, *ucfirst("cadena");*, *ucwords("cadena");*, que converteixen en majúscula el primer caràcter de la cadena indicada o de cadascuna de les paraules, respectivament, *strcasecmp("cadena1","cadena2");*, que compara (de forma case-sensitive) les dues cadenes indicades i retorna 0 si ambdues són iguals, -1 si la primera va abans alfabèticament que la segona i 1 si va després (la variant case-insensitive és *strcmp("cadena1","cadena2");*), etc. D'altra banda, també tenim les funcions de "hashing", com *sha1("cadena");*, que retorna el hash SHA1 de la cadena indicada, o *password_hash("cadena", algoritme);*, que retorna el hash amb sal aleatòria de la cadena indicada calculat amb l'algoritme indicat (o bé el corresponent a la constant *PASSWORD_DEFAULT*, que representa l'algoritme per defecte triat pel sistema), o *password_verify("cadena", "hash");*, que retorna *true* si la cadena indicada genera el hash indicat, etc. Teniu exemples d'aquestes funcions i moltes més a https://www.w3schools.com/php/php_ref_string.asp

> NOTA: Strings have one special usage that set them apart a little bit from the other variable types, and that is *{x}* notation. As a string is just a collection of characters, it is sometimes possible that you may want to read or write just one character. Take a look at this example, which outputs the "Hello, world!" sentence:
>
> ```
> <?php
>   $mystr = "Jello, world?";
>   $mystr{0} = "H";
>   $mystr{12} = "!";  //We could have done this equivalently: $mystr{strlen($mystr) - 1}="!";
>   print($mystr);
> ?>
> ```

> NOTA: The basic string operations, like concatenating two strings and assigning strings to variables, don't need anything special for UTF-8. However, most string functions, like *strpos()* and *strlen()*, do need special consideration. These functions often have an "mb_*" counterpart: for example, *mb_strpos()* and *mb_strlen()*. These "mb_*" strings are made available to you via the "Multibyte String Extension" (https://www.php.net/manual/en/book.mbstring.php) and are specifically designed to operate on Unicode strings. So you must use the "mb_*" functions whenever you operate on a Unicode string because if you use substr() on a UTF-8 string, there's a good chance the result will include some garbled half-characters.

* Relacionades amb les matemàtiques (https://www.php.net/manual/en/book.math.php): Aquí ens podem trobar funcions com *ceil(x.y);* *floor(x.y);* o *round(x.y);* que retornen el número sencer inmediatament superior, inferior o més proper, respectivament al número decimal indicat (opcionalment, *round()* pot admetre un segon paràmetre -sencer- que serveix per indicar fins a quin decimal es vol arribar amb l'arrodoniment -per defecte val 0, doncs-), o les corresponents a conversions trigonomètriques (les quals admeten valors decimals -representant radians- i retornen valors decimals també) com *sin(x.y);* *cos(x.y);* *tan(x.y);* així com les seves inverses *asin(x.y);* *acos(x.y);* o *atan(x.y);* etc, o *abs(x.y);* que retorna el valor decimal absolut del nombre decimal indicat, o *sqrt(x.y);* que retorna l'arrel quadrada del nombre decimal indicat, o *pow(x.y,z.a);* que retorna la potència $x.y^{z.a}$, o funcions de canvi de base com *decbin(x);*, que retorna una cadena binària corresponent al nombre decimal indicat, o *dechex(x);* que retorna el valor hexadecimal corresponent, o *hexdec(XX);*, que retorna el valor decimal corresponent al valor hexadecimal indicat o la més general *base_convert(nº,baseOriginal,baseNova);*, o max(array); i min(array); les quals retornen el valor màxim i mínim, respectivament, presents en l'array indicat, etc. D'altra banda, també tenim les funcions de generació de nombres sencers aleatoris, com, entre d'altres, *rand(x,y);* la qual admet com a paràmetres respectivament el nombre sencer mínim i màxim que formaran el rang -ambdós

inclosos- des d'on s'obtindrà, cada cop que s'executi, un nombre aleatori diferent. La documentació d'aquestes funcions aleatòries es troba a https://www.php.net/manual/en/book.random.php Finalment, també disposem de vàries constants matemàtiques predefinides, com ara *M_PI* (equivalent al nombre Pi), *M_PI_2* (equivalent a Pi/2), *M_1_PI* (equivalent a 1/Pi), *M_SQRT2* (equivalent a l'arrel quadrada de 2), etc. En tot cas, teniu exemples d'aquestes funcions, constants i molt més a https://www.w3schools.com/php/php_ref_math.asp

* Relacionades amb la gestió de dates (https://www.php.net/manual/en/book.datetime.php): Aquí ens podem trobar funcions com *time();*, que retorna el temps actual (en format "Unix time", és a dir, un nombre sencer -anomenat "timestamp"- que representa la quantitat de segons que han transcorregut des de 1-1-1970; aquest format és molt pràctic per realitzar operacions sobre dates com adicions o substraccions), o *strtotime("2023-09-26 14:39:56");*, que converteix la cadena indicada (en el format indicat, -tot i que hi ha d'altres possibles-) a nombre "timestamp" o retorna -1 si no pot fer-ho, o de forma similar *mktime(n,n,n,n,n,n);* que converteix a un únic valor "timestamp" el conjunt de nombres indicats (que en ordre representen, respectivament: hores, minuts, segons, mesos, dies, anys) , o al contrari, *date("Y-m-d H:i:s", n);*, que converteix el valor "timestamp" indicat com a segon paràmetre (o si no se l'indica cap, es prendrà el valor de l'instant actual) a una cadena amb un format concret (on diferents símbols representen diferents seccions de la cadena a retornar, que representa la data al complet). En tot cas, teniu exemples d'aquestes funcions, constants i molt més a https://www.php.net/manual/en/ref.datetime.php

> NOTA: El comportament de les funcions anteriors es veuran afectades pel valor que tingui la directiva de configuració de l'intèrpret PHP *date.timezone*, la qual indica la zona horària de referència de l'intèrpret PHP (si no s'estableix, per defecte s'utilitzarà UTC)
>
> NOTA: El llenguatge PHP proporciona una forma alternativa de gestiona les dates que és mitjançant la programació orientada a objectes (en lloc de la procedimental, que és la que estem estudiant en aquest document). Concretament, proporciona la classe "DateTime" que conté els mètodes i propietats necessàries per obtenir, modificar, comparar o calcular valors de data i hora, a més de gestió de calendaris i de zones horàries. En teniu la referència oficial completa a https://www.php.net/manual/en/class.datetime.php

* Relacionades amb la gestió del filesystem (https://www.php.net/manual/en/book.filesystem.php): Aquí ens podem trobar funcions com *file_get_contents("/ruta/arxiu");* que retorna el contingut de l'arxiu indicat en forma de cadena, o *file("/ruta/arxiu");* que retorna el contingut de l'arxiu indicat en forma d'array on cada element és una línia de l'arxiu en qüestió, o el conjunt de funcions *$fitxerID=fopen("/ruta/arxiu", X);* -on *X* pot ser "*r*" (per poder llegir), "*w*" (per poder escriure) o "*a*" (per per afegir), entre d'altres, i *$fitxerID* recollirà l'identificador de fitxer que s'utilitzarà a les següents funcions que hi treballaran sobre el fitxer en qüestió (o *false* si hi ha hagut algun error)...també podem fer servir *$fitxerID=tmpfile();*-, *$contingut=fread($fitxerID, n);* -on "n" representa la quantitat de bytes que es voldran llegir...si es vol llegir de cop tot el fitxer és util indicar aquí la funció *filesize($fitxerID);* i *$contingut* és el contingut llegit retornat-, *fwrite($fitxerID, "contingut");* que escriu el contingut indicat al fitxer indicat (sobreescrivint el contingut actual a bé afegint-lo a continuació de l'actual segons el mode d'obertura indicat a *fopen()*) i *fclose($fitxerID);* (entre altres com *fseek($fitxerID, posició);* i *rewind($fitxerID);* o *fgets($fitxerID);*, *fgetc($fitxerID);* i *feof($fitxerID);*, etc), o la funció *file_put_contents("/ruta/arxiu", "contingut");* que escriu el contingut indicat al fitxer indicat (sobreescrivint el contingut actual a no ser que s'indiqui com a tercer paràmetre la constant *FILE_APPEND*, que fa que el nou contingut s'escrigui a continuació de l'actual) i retorna el nombre de bytes escrits o *false* si hi ha hagut algun error. D'altra banda, també tenim funcions de gestió del sistema de fitxers com *rename("/ruta/fitxer1","/ruta/fitxer2");* -que serveix per moure també- *copy("/ruta/fitxer1","/ruta/fitxer2"); unlink("/ruta/fitxer");* -per esborrar-lo-, *file_exists("/ruta/fitxer");* -que retorna *true* si existeix el fitxer indicat i *false* si no-, *is_readable("/ruta/fitxer"); is_writeable("/ruta/fitxer"); is_executable("/ruta/fitxer"); is_file("/ruta/fitxer"); is_dir("/ruta/fitxer");* -per comprovar diferents característiques del fitxer/carpeta indicat-, *chmod("/ruta/fitxer", 0xxx);* -per establir els permisos indicats al fitxer/carpeta indicat-, *chown ("/ruta/fitxer","nomUsuari");* -per canviar el propietari-, *rmdir("/ruta/carpeta");* -elimina la carpeta indicada (ha d'estar buida, però, per a què funcioni)-, *scandir("/ruta/carpeta");* -retorna un array de cadenes on cada element val la ruta d'un fitxer o subcarpeta ubicat dins de la

carpeta indicada-, etc, etc. En tot cas, teniu exemples d'aquestes funcions, constants i molt més a https://www.php.net/manual/en/ref.filesystem.php

* Relacionades amb l'execució de programes externs (https://www.php.net/manual/en/book.exec.php): Aquí podem destacar la funció *exec("/ruta/comanda");* , la qual executa la comanda del sistema indicada i retorna la darrera línia de la seva sortida (se sol utilitzar d'aquesta manera quan la sortida completa ens és irrellevant) o *false* si hi ha hagut algun error; no obstant, si s'hi afegeix un segon paràmetre, el qual ha de ser de tipus array, la sortida de la comanda llavors es guardarà en aquest array, on cada element serà una línia diferent de la sortida. Una altra funció interessant és *shell_exec("/ruta/shellscript");*, la qual invoca un shell per executar-hi l'script indicat i retornar tota la seva sortida en forma de cadena (o *false* si hi ha hagut algun error), entre altres.

NOTA: Existeixen moltíssimes més funcions interessants, com ara les relacionades amb l'establiment de connexions de xarxa (veure https://www.php.net/manual/en/book.sockets.php), l'enviament de correus (https://www.php.net/manual/en/book.mail.php), el processament de contingut JSON (https://www.php.net/manual/en/book.json.php), el processament d'imatges (https://www.php.net/manual/es/book.image.php), funcions miscelània (https://www.php.net/manual/en/ref.misc.php) entre les quals podem destacar *sleep(n);* - http://www.hackingwithphp.com/4/11/0/pausing-script-execution-sleep-and-usleep – o *connection_status();* - http://www.hackingwithphp.com/4/13/0/connection-related-functions-ignore_user_abort-register_shutdown -,etc