Entorns de Desenvolupament

14. Funcions: declaració i crides

Unió Europea
Fons social europeu
L'FSE inverteix en el teu futur

IES DR. LLUÍS SIMARRO

14. Funcions: declaració i crides

Una de les eines més importants en qualsevol llenguatge de programació són les funcions. Una funció és un conjunt d'instruccions que al llarg del programa van a ser executades multitud de vegades. Per estalviar línies de codi, fent-lo més llegible, aquest conjunt d'instruccions s'agrupen en una funció que després podrà ser cridada i executada des de qualsevol punt del programa.

A més, una funció pot rebre paràmetres externs dels quals depenga el seu resultat. És a dir, segons el paràmetre o paràmetres amb els quals s'invoque la funció, aquesta retornarà un resultat o un altre.

Les funcions han d'estar **definides abans de realitzar la crida** a la funció (com és lògic). Totes les **funcions i classes de PHP tenen àmbit global**. Es poden cridar des de fora d'una funció fins i tot si van ser definides dins, i viceversa.

Sintaxi general per declarar una funció en PHP:

```
function nom (paràmetre1, paràmetre2, ..., paràmetreN) {
    instrucció1;
    instrucció2;
    ...
    instrucciónN;
}
```

Per cridar la funció (fer que s'execute) utilitzarem aquesta sintaxi, dins del codi del programa:

```
...
nom (par1, par2, par3, ..., parn);
...
```

on par1, par2, par3, ..., parn són els paràmetres (informació) que li passem a la funció.

Una funció pot necessitar de cap, un o diversos paràmetres per executar-se.

Exemple1:

```php
<?php // Exemple funció

    // Declaració de funcions
    function mostrarText($text) {
        echo "<strong> El text a mostrar és el següent: </strong>" ;
        echo $text;
        echo "<br/>";
    }
    // Fi de declaració de funcions

    // crida a la funció
    mostrarText("IES Simarro");
?>
```

En aquest exemple hem vist com hem definit una funció el nom de la qual és `mostrarText`. Aquesta funció espera un paràmetre quan és invocada (paràmetre que s'ha anomenat `$text`). Una vegada s'executa, la funció executa una sèrie d'instruccions i retorna el control al punt des del qual va ser invocada.

En alguns llenguatges de programació es distingeixen els termes "**procediment**" quan un fragment de codi d'aquest tipus executa una sèrie d'instruccions **sense retornar un valor**, enfront del terme "**funció**" que s'aplica quan un fragment de codi d'aquest tipus executa una sèrie d'instruccions i **retorna un valor**. En **PHP no es distingeix** entre una cosa i una altra, simplement es parla de "**funció**" en general.

En alguns llenguatges de programació com Java l'especificació de tipus que es van a rebre per part de la funció (o el tipus de dades que va a tornar la funció) és molt més forta. Si ens fixem, la funció `mostrarText` rep un paràmetre anomenat `$text`, però en cap

lloc s'especifica si aquest paràmetre és tipus `integer`, `float`, `double` o `string`. De quin tipus és? Realment no ho sabem: l'intèrpret PHP s'encarrega automàticament de reconèixer el tipus que li passa a la funció. A més, intentarà executar el codi siga com siga el tipus del paràmetre passat. Si li resultara impossible executar el codi, **tornaria un error**.

La utilitat fonamental de les funcions és **no haver de repetir parts de codi comunes**, que seria necessari escriure diverses vegades. Aquestes parts de codi comunes s'agrupen en funcions i simplement cridarem a la funció indicant els paràmetres necessaris cada vegada que necessitem executar aquest codi. D'aquesta manera, evitem la repetició que **fa més llarg i difícil d'entendre un programa** o desenvolupament web.

També podem crear funcions que retornen dades (valors concrets). Aquestes funcions, que podríem denominar "**funcions en sentit estricte**", són aquelles que executen un codi i com a punt final d'aquest codi inclouen una sentència `return` seguida del resultat de la funció. La sentència `return` indica que s'ha arribat al final de la funció i es torna com a resultat de la mateixa el contingut especificat a continuació del `return`. Després d'un `return` pot retornar una variable, un nombre, una cadena de text, etc.

Per **exemple**:

```
return "No disposa de permisos";
```

vol dir que la funció retorna aquesta cadena de text.

Un altre **exemple**:

```
return $calcul;
```

indica que la funció retorna el contingut que es trobe emmagatzemat en la variable `$calcul`.

Un altre **exemple**:

```
return "Malauradament ".$usuari." no disposa de permisos. Per a
 sol·licitar informació pot escriure a ".$emailAdministrador;
```

faria que la funció retornara una cadena de text on intervenen diverses variables.

Exemple 2:

```
<?php //Exemple funcions
      //inici declaració de funcions
      function operacions ($n1, $n2, $operacio) {
            $resultat = 0;
            if ($operacio == "Sumar" ) {
                  $resultat = $n1 + $n2;
            } else if ($operacio == "Restar" ) {
                  $resultat = $n1 - $n2;
            } else if ($operacio == "Multiplicar" ) {
                  $resultat = $n1 * $n2;
            }
      return $resultat; // Tornar el resultat final en qualsevol cas
      }
      //final de la declaració de funcions

      // Crida a la funció operacions:
      //podem cridar i després mostrar per pantalla
      $valorResultat = operacions (5, 7, "Sumar" );
      echo  "Resultat: ".$valorResultat."<br/>" ;
      // O podem mostrar per pantalla directament
      echo "Resultat: ".operacions(15, 8, "Restar" )."<br/>" ;
?>
```

A diferència de la funció `mostrarText`, la funció operacions ens retorna un valor concret, de manera que **se substitueix la seua invocació allà on apareix pel valor que retorna**. Així, la instrucció:

```
echo operacions (15, 8, "Restar");
```

equivaldria a escriure:

```
echo 'valor retornat per la funció operacions invocada amb paràmetres 15, 8 i "Restar"';
```

és a dir, seria el mateix que escriure:

```
echo (15-8);
```

o

```
echo 7;
```

A més, observem que la funció **mostrarText** requeria un paràmetre, mentre que la funció operacions requereix tres paràmetres. Si s'invoca la funció **sense** passar-li el **nombre** de **paràmetres adequat** s'obtindrà un error del tipus:

"Warning: Missing argument 3 for operacions () "

Finalment, hem tenir en compte que una funció pot ser també invocada sense paràmetres.

Exemple 3:

```
<?php // Exemple funcions
      function mostrarTextError {
            echo "<strong> S'ha produït un error </strong><br/>" ;
      }
?>
```

Aquesta funció no té paràmetres. Per invocar, escriuríem simplement:

```
mostrarTextError();
```

Cada vegada que realitzàrem la invocació s'executaria el codi dins de la funció. Aquesta funció podem dir que és "**tipus procediment**" perquè **no retorna un resultat** (no té sentència **return**).

PHP **no admet la sobrecàrrega de funcions**, **ni** és possible '**desdefinir**' ni **redefinir funcions** prèviament declarades.

Els noms de les **funcions** són **insensibles** a **majúscules-minúscules**, encara que és

una bona idea cridar a les funcions tal com apareixen en les seues declaracions.

En PHP és possible cridar a funcions **recursives**.

Exemple 4:

```
<?php
    function recursivitat($a){
        if ($a < 20) {
            echo "$a <br/>";
            recursivitat($a + 1);
        }
    }
?>
```

Però s'ha de tenir en compte que les crides a funcions recursives amb més de 100-200 nivells de recursivitat poden **esgotar la pila** i ocasionar la finalització de l'script en curs. Especialment, les **recursivitats infinites** estan considerades un **error greu de programació**.