

Hoofdstuk 7 : Functies

7.1 : Definitie

Een functie is een ‘mini-programma’. Bij het schrijven van PHP code, zijn er vaak bewerkingen die steeds terug komen. We kunnen die bewerking definiëren in een functie (function). Vervolgens zullen we ergens in de code, de gedefinieerde functie oproepen. Zo besparen we ons zelf de moeite om steeds dat stukje code opnieuw te schrijven.

Als u dus in een script een bepaalde taak meer dan 1 keer uitvoert (een berekening, een gegeven uit een database halen,...) is het interessant om voor die bepaalde taak een functie te schrijven die het uitvoeren van die taak op zich neemt.

We geven iedere functie een specifieke naam. De functie wordt uitgevoerd door die naam aan te roepen.

Als PHP programmeur kun je zelf zoveel functies schrijven als je wil. Daartegenover bestaan er ingebouwde functies (standaard functiegroepen) die rechtstreeks deel uit maken van de programmeertaal PHP.

Een mooi overzicht staat op <http://www.php.net/quickref.php>



De basis structuur voor het maken van een functie is:

```
function functienaam ($argument1, $argument2,...) {
  ..//hier staan de bewerkingen die we willen definiëren in de functie
}
```

We kunnen vervolgens de functie later in de code oproepen door zijn naam ‘*functienaam*’

Om mogelijke verwarring te vermijden adviseer ik om steeds kleine letters te gebruiken. (ook al is de 'functienaam' niet hoofdletter gevoelig).

Gebruik ook geen spaties bij functienamen. Gebruik een underscore ('_')

Voorbeeld:

btw_inclusief(\$a)

7.2 Oefening

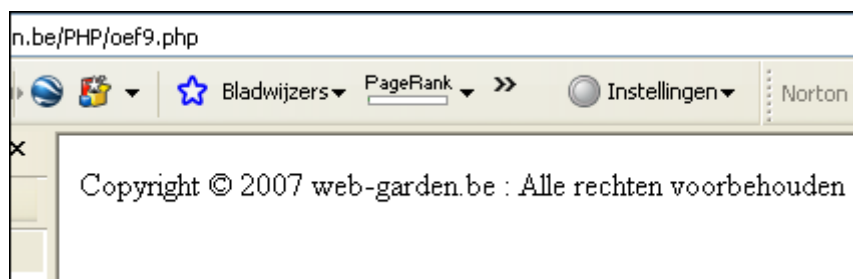
Open start.php en voeg volgende code toe.

```
<?php
function voettekst(){
echo ("Copyright &copy; " .date ("Y"). " web-garden.be : Alle rechten voorbehouden");}
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>cursus PHP - www.web-garden.be</title>
</head>

<body>
<?php
voettekst ();
?>
</body>
</html>
```

Sla op als **oef9.php**. Via FTP kun je dit bestand downloaden in de map PHP op je webserver. Open vervolgens dit bestand via je browser.

Dit is wat je verkrijgt:



Laten we de code even van dichterbij bekijken.

De pagina begint met een stukje PHP code waar we de functie met naam voettekst aanmaken.

```
<?php
function voettekst(){
echo ("Copyright &copy; " .date ("Y"). " web-garden.be : Alle rechten voorbehouden");}
?>
```

Het feit dat we beginnen met de aanheft 'function' betekent dat we hier een functie gaan definiëren. De aanheft 'function' wordt gevolgd door de naam die we geven aan de functie...in ons voorbeeld:voettekst.

De naam van de functie (in ons geval 'voettekst'), wordt steeds gevolgd door 2 haakjes ('()').

In ons voorbeeld staat er geen 'argument' tussen de haakjes.

Vervolgens gaan we definiëren wat de functie 'voettekst' dient uit te voeren. Dit alles wordt geplaatst tussen 2 accolades.

De actie die er moet uitgevoerd worden is een 'echo' statement.

Een stukje tekst waarin het actuele jaartal (date("Y")) wordt geplaatst.

De code '©' is de tekst weergave van het symbool ©.

Het is u waarschijnlijk opgevallen dat de plaats van deze PHP code nogal vreemd is.

De code staat namelijk vóór het DOCTYPE statement.

Voor een goede leesbaarheid van de code, zullen veel programmeurs de functie-definities plaatsen voor de HTML code.

In de <body> van de webpagina wordt de functie voettekst nu aangeroepen.

```
<?php
voettekst ();
?>
```

Hier wordt gewoon de naam van de functie weergegeven, weerom gevolgd door 2 haakjes. De functie-oproep wordt afgesloten met een punt-komma (;)

7.3 Argumenten in de functie.

In bovenstaand voorbeeld hebben we een functie 'voettekst' gemaakt zonder 'argumenten' tussen de haakjes.

We geven hier nu een voorbeeld waarbij we wel gebruik maken van een argument.

Het betreft de berekening van een verkoopprijs , op basis van de inkoopprijs.

Open terug start.php, en voeg onderstaande code toe in de <body>.

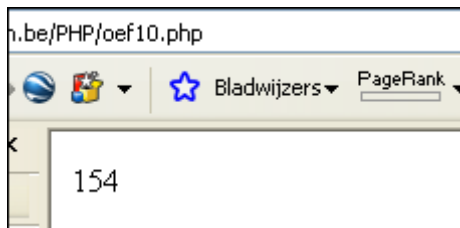
```
<?php
function berekening($num){
$num = ($num / 0.65);
$num = round($num,0);
return $num;}

$inkoopprijs = 100;
$verkoopprijs = berekening($inkoopprijs);
echo ("$verkoopprijs");
?>
```

Sla alles op onder oef10.php. Via FTP kun je dit bestand vervolgens downloaden in de map PHP op de server.

Open vervolgens de file via je browser.

Dit is het resultaat die je ziet.



De code vraagt natuurlijk wat uitleg.

De code begint met het definiëren van de functie 'berekening'.

```
function berekening($num){
$num = ($num / 0.65);
$num = round($num,0);
return $num;}
```

De aanheft 'function' heeft aan dat we hier een functie zullen definiëren.

We geven de functie de naam 'berekening'.

De naam wordt gevolgd door 2 haakjes met daartussen het 'argument' (\$num).

Dit betekent dat als we de functie later oproepen, dat bij de oproep er een waarde tussen de haakjes moet staan.

Die waarde (argument) zal gebruikt worden bij de bewerking binnen de functie-omschrijving.

Het 'argument' \$num zal dus gebruikt worden bij de bewerkingen die binnen de functie worden omschreven.

We gebruiken hier voor het eerst de statement 'round'. Dit betekent dat de variabele \$num binnen de haakjes van het 'round' statement afronden, met '0' cijfers na de komma.

Met de statement return geven we aan de functie de uiteindelijke waarde van \$num.

Nu bekijken we de rest van de PHP code.

```
$inkoopprijs = 100;
$verkoopprijs = berekening($inkoopprijs);
echo ("$verkoopprijs");
```

Eerst krijgt de variabele \$inkoopprijs de waarde 100 mee.

Vervolgens wordt de waarde van de variabele \$verkoopprijs berekend door de functie 'berekening' los te laten op het argument '\$inkoopprijs'.

Maw als we nu even terug kijken naar de code van de functieomschrijving, komt dus de variabele \$inkoopprijs i.p.v. \$num.

Zoals reeds eerder aangegeven verwacht de functie 'berekening' een argument(variabele) tussen de haakjes.

Die variabele (argument) zal de functie berekening oppikken als het argument \$num, en gebruiken in de verdere bewerkingen.

7.4 Meerdere argumenten in de functie

Het voorbeeld in paragraaf 7.3 had bij de functie-omschrijving, 1 argument staan tussen de haakjes:

```
function berekening($num)
```

We hebben toen aangegeven dat bij de aanroep van de functie er een argument (variabele) MOET staan tussen de haakjes

```
berekening($inkoopprijs)
```

'\$num' noemt me ook wel eens het vereiste argument.

We hebben in paragraaf 7.1 de definitie van een functie als volgt weergegeven:

```
function functienaam ($argument1, $argument2,...) {
  ..//hier staan de bewerkingen die we willen definiëren in de functie
}
```

\$argument1 is dus het vereiste argument, m.a.w. bij de aanroep van de functie moet er argument staan tussen de haakjes.

\$argument2, \$argument3,... zijn vrijblijvende argumenten.

Ze kunnen in de functie-omschrijving (functie-definitie) worden opgegeven, maar ze moeten niet aanwezig zijn bij de functie-aanroep.

Laat ons dit illustreren met een voorbeeld.

In ons laatste voorbeeld hebben we de verkoopprijs berekent, door de inkooprijs te delen door 0.65.

Het kan natuurlijk best zijn dat we niet altijd de verkoopprijs willen berekenen met de coëfficiënt 0.65

Open start.php, en voeg onderstaande code toe in de <body>

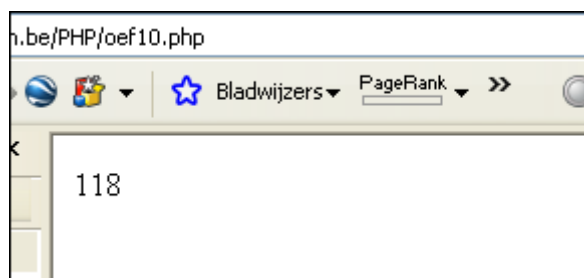
```
<?php
function berekening($num, $num2){
    $num = ($num / (0.65+$num2));
    $num = round($num,0);
    return $num;}

$inkooprijs = 100;
$verkoopprijs = berekening($inkooprijs,0.2);
echo ("$verkoopprijs");
?>
```

Sla dit op als **oef11.php**. Via FTP kun je dit bestand downloaden naar de map PHP op je server.

Open vervolgens dit bestand in je browser (<http://www.mijnsite.com/PHP/oef11.php>)

Dit is het resultaat:



Zoals steeds vraagt deze code wat uitleg.

We bekijken weerom eerst de code die gebruikt wordt voor het definiëren van de functie.

```
function berekening($num, $num2){
    $num = ($num / (0.65+$num2));
    $num = round($num,0);
    return $num;}
```

De aanhef 'function' betekent dat we te maken hebben met het opmaken van een functie. De naam van de functie is 'berekening'.

In vergelijking met het voorbeeld in paragraaf 7.3, hebben we nu 2 argumenten tussen de haakjes.

\$num is het vereiste argument. M.a.w. die moeten we straks terug vinden in de aanroep van de functie.

\$num2 is de vrijblijvende variabele. Die wordt gebruikt in de beschrijving van de functie, maar moet niet worden gebruikt in de aanroep van de functie.

De rest van de code uit de functie-beschrijving is dezelfde als in paragraaf 7.3, met uitzondering van de formule voor de berekening van \$num.

```
$num = ($num / (0.65+$num2));
```

We bekijken nu de rest van de code:

```
$inkoopprijs = 100;  
$verkoopprijs = berekening($inkoopprijs,0.2);  
echo ("Verkoopprijs");
```

Bij de berekening van \$verkoopprijs, roepen we de functie 'berekening' op.

Tussen de 2 haakjes staan nu 2 argumenten (gescheiden door een komma).

De eerste (\$inkoopprijs) is de vereiste variabele. Zonder opgave van deze kan er verder niets gebeuren.

De tweede variabele hebben we hier de waarde 0.2 meegegeven.

Dus in de functie-verwerking, krijgt \$num2 de waarde 0.2

Als we het tweede argument (0.2) niet opgeven bij de aanroep...dan zal de verkoopprijs worden berekend als inkoopprijs/0.65

7.5 Het keyword 'Global'

Laat ons even direct met de deur in huis vallen.

Open start.php, en voeg onderstaande code toe in de <body>.

Sla op in **oef12.php**, en download via FTP in de map PHP op je server.

```
<?php  
function global_test(){  
$getal=20;  
echo ("Hier komt " . $getal. " te staan");}  
$getal=50;  
global_test();  
?>
```

Dit heeft volgend resultaat in de browser:



Even de code van naderbij bekijken.

We maken een functie met de naam 'global_test'.

Daarin krijgt de variabele \$getal de waarde 20 mee.

Vervolgens hebben we een 'echo' statement waarin de variabele \$getal wordt verwerkt.

Op zich niets nieuws onder de zon☺

Vervolgens gaan we, NA de functie-definitie, een waarde van 50 toekennen aan de variabele \$getal.

Dan roepen we de functie 'global_test' aan.

De uitkomst op het scherm zie je hierboven.

Is dit fout? Want de laatste waarde die we aan \$getal hebben gegeven is 50...en toch wordt de waarde van 20 gebruikt zoals bepaald in de functie-definitie.

Wat er weergegeven wordt is correct.

Wat er tussen de accolades staat behoort specifiek toe tot de functie-definitie. Het veranderen van variabelen BUITEN de accolades heeft geen invloed op wat er binnen de haakjes staat orf gebeurt.

In feite hebben we hier dus 2 variabelen met de naam \$getal.

Door het keyword 'global' te gebruiken kunnen we binnen een PHP-functie aangeven dat een waarde van een variabele willen gebruiken die ergens anders is gedefinieerd.

We bewerken de bovenstaande code als volgt.

```
<?php
function global_test(){
    global $getal;
    echo ("Hier komt " . $getal. " te staan");
    $getal=50;
    global_test();
?>
```

We krijgen nu volgend resultaat in de browser:



Binnen de functie-beschrijving, geven we d.m.v.

```
global $getal;
```

aan dat we de variabele \$getal gebruiken, die ergens BUITEN de functie-definitie is weergegeven (in ons voorbeeld wordt dit de waarde 50)

PS: vergeet na het gebruik van global geen punt-komma (;) te plaatsen !

7.6 Static variabelen

Open start.php, en breng volgende code in de <body>

```
<?php
function static_test(){
    $getal =1;
    $getal++;
    echo ("De waarde " . $getal. " wordt weergegeven <br>");
}
for ($i=1; $i<=5;$i++){
    static_test();}
?>
```

Sla alles op als **oef13.php**. D.m.v. FTP kun je dit bestand downloaden naar de map PHP op de server.

Op het opgeladen bestand via je browser. (<http://www.mijnsite.com/PHP/oef13.php>)

Dit is het resultaat in de browser.



Binnen de functie-definitie geven we aan de variabele \$getal de waarde 1. Vervolgens wordt er bij de variabele \$getal, 1 bijgeteld (\$getal++) Dit betekent dat de variabele \$getal dan inderdaad de waarde 2 heeft.

Dan starten we met 'for' een lus, bestaande uit 5 stappen (\$i<=5)

Als de lus aan zijn tweede stap is, kun je misschien verwachten dat er staat:

De waarde 3 wordt weergegeven

Zoals aangegeven is in de eerste stap de waarde van de variabele \$getal uiteindelijk gelijk aan 2.

Wanneer de functie 'static_test' voor de tweede keer in de lus wordt aangeroepen, wordt de waarde van de variabele \$getal weerom gelijk gesteld aan 1, om er dan met '\$getal++' , 1 bij op te tellen.

Waardoor de output op het scherm voor de tweede stap in de lus weerom

De waarde 2 wordt weergegeven

Laat ons bovenstaande code even vervangen door:

```
<?php
function static_test(){
static $getal =1;
$getal++;
echo ("De waarde " . $getal. " wordt weergegeven <br>");
}
for ($i=1; $i<=5;$i++){
static_test();}
?>
```

Met deze code krijgen we nu volgend resultaat op het scherm:



In vergelijking met de oorspronkelijke code, is er slechts 1 lijn gewijzigd:

```
static $getal =1;
```

Vóór de variabele \$getal staat nu het keyword 'static'.

Door de vermelding van 'static' zal PHP de waarde van de variabele \$getal onthouden, als de functie voor de tweede keer wordt aangeroepen in de lus.

Bij de tweede aanroep in de lus ('for'), weet PHP dat de laatste waarde van \$getal gelijk was aan 2.

D.m.v. \$getal++ telt hij er 1 bij op, waardoor er via de 'echo'

De waarde 3 wordt weergegeven

komt te staan.

Bij de derde aanroep in de lus, weet PHP (via het keyword static) dat de laatste waarde van \$getal, 3 was.

Via \$++, wordt er 1 bijgeteld, waardoor er via 'echo'

De waarde 4 wordt weergegeven

komt te staan.....