PHP

programmering

Udarbejdet af Benny Dyhr Thomsen

INDHOLD

PHP

programmering

Du kan bruge alm. html koder i PHP programmer. Når du vil bruge PHP-kode skal filtypen være **.php** i stedet for **.htm** eller **.html**.

PHP fundamentalt

Statements

PHP-kode kan lægges imellem html-koder, PHP-kode indledes med taggen **<?php** og afsluttes med **?>.**

```
<HTML>
<HEAD>
</HEAD>
<BODY>
to plus tre er <?php echo(2 + 3); ?>
</BODY>
</HTML>
```

- Source 1

Hver sætning afsluttes med semikolon.

```
<?php
echo(2 + 3);
echo(3 * 2);
echo("Goddag");
?>
```

- Source 2

Syntaksen er uafhængig af opstilling.

```
<?php
echo(2
+
3); echo(3
* 2
); echo(
      "Goddag")         ;
?>
```

- Source 3

code block { } samler flere statements til ét.

```
<?php
if (3 > 2) {
    echo("Goddag");
    echo(2 + 3);
}
?>
```


Benny Dyhr Thomsen

- Source 4

```
<?php
if (3 > 2) {
    echo("Goddag");
?>
Dette er html output og fortolkes derfor ikke af PHP-
  fortolkeren.
<?php
    echo(2 + 3);
}
?>
```

- Source 5

Kommentarer

```
<?php
echo(”Klik på OK”); // Meddelelse til brugeren
?>
```

- Source 6

```
<?php
echo(”Klik på OK”); # Meddelelse til brugeren
?>
```

- Source 7

```
<?php
// udskriv udregning
echo(2    //første tal
     +    //addition
     3    # andet tal
);
# udregning udskrevet
?>
```

- Source 8

```
$s = 600;
$n = 3;
/* Her beregnes gennemstilig salg for sælgere ved
   at dividere samletsalg $s med antal sælgere $n,
   resultatet gemmes i $gennemsnit */
$gennemsnit = $s/$n;
echo($gennemsnit);
```

- Source 9

```
<?php
echo(”Klik på OK”); # Meddelelse til brugeren
?>

// Denne tekst vises i browseren – ups!

<!-- Denne tekst vises ikke fordi den er en Html-kommentar
```

```
-->
```

- Source 10

Literaler

Du findes 3 typer af literaler tekst-literaler (string), numeriske (decimal og integer) og booleske (true og false)

Tekst-literaler

Når du putter tekst i dobbelt kvoter fortolker PHP specielle tegn. Den kigger efter variabel navne og erstatter dem med deres værdier. Den kigger efter backslash og den eller de følgende tegn for at finde specielle erstatninger.

Tegn efter backslash	Beskrivelse
n	Linefeed (LF)
r	Carriage return (CR)
t	Tab
\	Backslash
$	Dollar
”	Dobbelt kvote
Et octal nummer optil 3 cifre	En vilkårlig ACSII kode
x efterfulgt af hexcifre optil to cifre	En vilkårlig ACSII kode

Kode

```
<?php
echo(”Denne tekst\ngår over flere\nlinier\n\t\”og denne
   kvotering er indrykket\””);
?>
```

Output

```
Denne tekst
går over flere
linier
        ”og denne kvotering er indrykket”
```

- Source 11

Læg mærke til at browseren er ligeglad og viser linien som

```
Denne tekst går over flere linier ”og denne kvotering er indrykket”
```

Brug Vis, Kilde i browseren for at se outputtet.

Hvis du bruger enkeltkvote fortolkes linien ikke

Kode

```
<?php
echo('Denne tekst\ngår over flere\nlinier\n\t\"og denne
   kvotering er indrykket\"');
?>
```

Output

```
Denne tekst\ngår over flere\nlinier\n\t\"og denne kvotering er indrykket\"
```

- Source 12

Here documents

En måde at håndtere output på, bedst anvendt til større tekster ses her:

```
<?php
$kunde_navn = "Hansen";
    $ud=<<<slutkode
    <TABLE>
      <TR>
        <TD>Kunde navn</TD>
        <TD>$kunde_navn</TD>
      </TR>
    </TABLE>
slutkode;
    echo($ud);
?>
```

- Source 13

Bemærk at efter <<< vælger du et eller andet navn, her **slutkode**, som også skal afslutte tekstblokken. Det valgte navn + semikolon stå helt til venstremargenen på en linien efter tekstblokken.

Numeriske literaler

```
<?php
echo(255);   // Decimal
echo(0xFF);  // Hex
echo(0377);  // Octal
echo(0.0023);
echo(2.3e-3);   // 0.0023
echo(-3.65e6);  // -3.650.000
?>
```

- Source 14

Trods alt bliver alle tallene vist som decimal uanset om de er indtastet som hex eller octal.

Variabler

Variable skal indledes med dollartegn ($), derefter kan du bruge alle alfanumeriske tegn og underscore dog skal det første være et bogstav. Variabler er case-sentitive. Følgende er vist 5 forskellige variabler:

```
$a
$b2
$kunde_navn
```

```
$et_langt_variabelnavn
$Kunde_navn
```

- Source 15

Assignments (tildelinger)

```
<?php
$a = ”Hallo”;
echo($a);
?>
```

- Source 16

```
<?php
$a = ”Hallo”;
echo(“$a verden!”);
?>
```

- Source 17

```
<?php
$a = ”Hallo”;
$b = 4;
$c = $a;
echo(“$c koen har $b ben”);
?>
```

- Source 18

```
<?php
$a = ”Hallo”;
$b = “$a verden!”;
$a = “Farvel”
echo($b);
?>
```

- Source 19

Reference

Variabler kan refereres indirekte. En reference har formen:

```
${expression}
```

Den refererer til variablen hvis navn er resultatet af expression. Betragt følgende udtryk:

```
${”kunde_navn”}
```

refererer således til variablen $kunde_navn.

Følgende refererer også til variablen $kunde_navn:

```
<?php
$kunde_navn = ”Hansen”;
$a = “navn”;
echo(${”kunde_$a”});
?>
```

- Source 20

```
<?php
$kunde_navn = ”Hansen”;
$a = ”navn”;

    $ud=<<<slutkode
    <TABLE>
      <TR>
        <TD>Kunde $a</TD>
        <TD>${”kunde_$a”}</TD>
      </TR>
    </TABLE>
slutkode;
echo($ud);

$kunde_tlf = ”99-112233”;
$a = ”tlf”;

    $ud=<<<slutkode
    <TABLE>
      <TR>
        <TD>Kunde $a</TD>
        <TD>${”kunde_$a”}</TD>
      </TR>
    </TABLE>

slutkode;
echo($ud);
?>
```

- Source 21

Hvis vi antager $a = “kunde_navn” refererer følgende også til variablen $kunde_navn:

```
${$a}
```

og den kan skrives på en kortere syntaks som:

```
$$a
```

Constants

Konstanter erklæres:

```
define(”JULEAFTEN”,”24 december”);
```

bruges **uden** $ da det ikke er en variabel

```
echo(JULEAFTEN);
```

```
<?php
if (defined(“JULEAFTEN”)) {
    echo(“JULEAFTEN er defineret”);
} else {
    echo(“JULEAFTEN er ikke defineret”);
}
?>
```

- Source 22

Data typer

Otte datatyper

- string
- integer
- double
- array
- boolean
- object
- resource
- unknown

Kode
<?php $a = “en tekst”; echo(gettype($a)); ?>
Output

```
string
```

- Source 23

```
<?php
$a = “2”;
echo(gettype($a));
settype($a, integer);
echo(gettype($a));
echo($a);
?>
```

- Source 24

Type casting

PHP har også type casting operatorer, hvilket giver dig mulighed for at behandle en værdi af en type som om det var en anden. Følgende kan bruges i casting

- (string)
- (integer) eller (int)
- (double)
- (boolean) eller (bool)

Bruges som følger

```
<?php
$a = “234.56”;
echo((integer)$a);
?>
```

- Source 25

```
<?php
$a = “2 kaffe tak”;
echo(gettype($a));
settype($a, integer);
echo(gettype($a));
echo($a);
?>
```

- Source 26

```
<?php
$a = “2 kaffe tak”;
$b = 3;
echo($a + $b);
?>
```

- Source 27

Operators og functions

Vi har set function kald brugt på denne måde:

```
echo(gettype(“hallo”));
```

Functions kaldet gettype(“hallo”) giver værdien “string”. Functions kald kan bruges alle steder hvor der bruges literale. Følgende er legal kode:

```
<?php
${gettype(“hallo”)} = “verden”;
echo(gettype(gettype(“hallo”)));
echo(“<BR>”>
echo($string);
?>
```

- Source 28

En operation er et udtryk indeholdende en operator.

```
echo(2 + 3);
```

en additionsoperator (+) er en binær operator, den opererer på to værdier. Andre operatorer er f.eks. relationsoperatoren (>), string sammenføjningsoperatoren (.) og tildelingsoperatoren (=).

Hvis additionen blev lavet med en function (findes ikke) i stedet for en operator, ville udtrykket se således ud:

```
echo(add(2, 3));
```

Betingelsesoperator

```
$a ? $b : $c;
```

```
<?php
define(”JULEAFTEN”,”24 december”);
if (defined(“JULEAFTEN”)) {
    echo(“JULEAFTEN er defineret”);
} else {
    echo(“JULEAFTEN er ikke defineret”);
}
// du kan gøre dette i stedet
echo(“JULEAFTEN er ” . (defined(”JULEAFTEN”) ? ”defineret”
   : ”ikke defineret”));
?>
```

- Source 29

Generelle operationer

```
<?php
echo($a = ”hallo”);
$b = $c = $a;
echo($b);
echo($c);
?>
```

- Source 30

Relationsoperatoren == evaluerer true hvis begge udtryk på hver side er ens. != giver false.

```
<?php
$a = 4;
echo($a == 4);
?>
```

- Source 31

```
<?php
$a = 1.2;
$b = 0.7;
$c = $a - $b;
echo(($c == 0.5) ? “sand” : “falsk”);
?>
```

- Source 32

String operators

PHP bruger operatoren (.) til at sammenføje strenge.

```
<?php
$a = “hallo”;
$b = “verden”;
$c = $a . $b;
echo($c);
?>
```

- Source 33

du kan bruge:

```
$a .= $b;
```

i stedet for:

```
$a = $a . $b;
```

```
<?php
$a = “hallo”;
$b = “verden”;
$c = “<b>” . $a . “ “ . $b . “</b>”;
echo($c);
?>
```

- Source 34

String functions

PHP tilbyder mange string-functions.

substr()

```
string substr(string haystack, int needle [, int
   length])
```

```
<?php
$s1 = substr(“Den kat er sød”, 4,3);
$s2 = substr(“Den kat er sød”, 0,1);
$s3 = substr(“Den kat er sød”, 8);
echo($s1 . ”<BR>”);
echo($s2 . ”<BR>”);
echo($s3 . ”<BR>”);
?>
```

- Source 35

strpos()

```
int strpos(string s, string seek [, int offset]);
```

```
<?php
$i1 = strpos(“Den kat er sød”,”kat”);
$i2 = strpos(“Den kat er sød så sød”,”sød”,13);
echo($i1 . ”<BR>”);
echo($i2 . ”<BR>”);
?>
```

- Source 36

htmlspecialchars()

```
string htmlspecialchars(string s [, int quote_style [,
   string charset]])
```

```
<?php
echo(htmlspecialchars(”<Dig & mig>”));
?>
```

- Source 37

trim()

```
string trim(string s)
```

```
<?php
echo(trim(”    Den kat er sød    ”));
?>
```

- Source 38

chr() og ord()

```
<?php
```

```
echo(chr(64));
echo(ord('@'));
?>
```

- Source 39

strlen()

```
int strlen(string s)
```

```
<?php
echo(strlen("en"));
echo(strlen("Den kat er sød"));
?>
```

- Source 40

printf() og sprintf()

```
int printf(string format [, mixed args...])
string sprintf(string format [, mixed args...])
```

Hvis **format** indeholdes to *conversions specifier* skal der også bruges to **args**.

En *conversion specifier* er %-tegnet efterfulgt af op til 5 specifiers i følgende orden:

- **Padding specifier**
  Udfyldningstegn hvis ingen angivet bruges space
- **Alignment specifier**
  Tegnet – venstrejusteret, hvis undladt højrejusteres.
- **Minimum width specifier**
  Min. bredde inkl. decimaler og decimalpunktum.
- **Precision specifier**
  Antal decimaler

- **Type specifier**

Type specifier	Beskrivelse
b	integer præsenteret som et binært tal
c	integer præsenteret som ASCII tegn
d	integer præsenteret som et decimal nummer
f	flydende decimal tal præsenteret som decimal nummer med decimal komma (punktum)
o	integer præsenteret som octal tal
s	string
x	integer præsenteret som hex tal i lowercase
X	integer præsenteret som hex tal i uppercase

```
<?php
$dag = 24;
$mnd = 12;
$aar = 2002;
printf(”%02d/%02d/%04d<BR>”,$dag,$mnd,$aar);
$salg = 123.5;
$s = sprintf(”salg er %.2f<BR>”,$salg);
echo($s);
?>
```

- Source 41

Numeriske operators

Operator	Beskrivelse
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulus

prefix-operator er foran som fortegn men også som vækstoperator

```
$a = -2; // fortegn
++$a; //vækstoperator øg
--$a; //vækstoperator mindsk
```

postfix-operator er bagved som vækstoperator.

```
$a = -2; // fortegn
$a++; //vækstoperator øg
```

```
$a--; //vækstoperator mindsk
```

Udtryk	Beskrivelse
++$a	$a øges med 1 og tallet udlæses
--$a	$a minskes med 1 og tallet udlæses
$a++	Tallet udlæses og $a øges med 1
$a--	Tallet udlæses og $a minskes med 1

Bit-operatorer

Operator	Brug	Beskrivelse
&	$a & $b	AND
|	$a | $b	OR
~	~$a	NOT
^	$a ^ $b	XOR
<<	$a << $b	left shift
>>	$a >> $b	right shift

```
<?php
define(“CREATE”,1);
define(“INSERT”,2);
define(“UPDATE”,4);
define(“DELETE”,8);

$permission = CREATE | DELETE;

echo(($permission & CREATE) ? “Permission: Create” : “”);
echo(($permission & INSERT) ? “Permission: Insert” : “”);
echo(($permission & UPDATE) ? “Permission: Update” : “”);
echo(($permission & DELETE) ? “Permission: Delete” : “”);

$permission |= INSERT;

echo(($permission & CREATE) ? “Permission: Create” : “”);
echo(($permission & INSERT) ? “Permission: Insert” : “”);
echo(($permission & UPDATE) ? “Permission: Update” : “”);
echo(($permission & DELETE) ? “Permission: Delete” : “”);
?>
```

- Source 42

```
<?php
define(“BIT2”,4);
echo(BIT2 << 4);
echo(“<br>”);
echo(BIT2 >> 2);
?>
```

- Source 43

Relations operatorer

Operator	Beskrivelse
>	Større end
>=	Større end eller lig med
<	Mindre end

<=	Mindre end eller lig med
==	lig med
===	identisk, både lig med og af samme type
!=	forskellig fra
!==	ikke identisk, enten ikke lig med eller ikke af samme type

```
<?php
echo((12 > 5));
echo(“<br>”);
echo((3.4e47 <= 3.8e17));
echo(“<br>”);
echo((23.5 != 23.5));
echo(“<br>”);
echo((‘a’ <= ‘b’));
echo(“<br>”);
echo((‘b’ < ‘B’));
echo(“<br>”);
echo((‘C’ == 67));
echo(“<br>”);
echo((‘C’ === 67));
echo(“<br>”);
echo((“Hansen” >= “Pedersen”));
echo(“<br>”);
?>
```

- Source 44

Operator prioritet

operator	**Prioritet**	**Beskrivelse**
+,-,++,--,~,casting operator	1	+ og – som fortegn
*,/,%	2	
+,-	3	til addition og subtraktion
<,<=,>,>=	4	
==,!=	5	
&	6	
^	7	
|	8	

```
<?php
$sum = 5 + 3 * 6;
echo($sum);
echo(“<br>”);
$sum = (5 + 3) * 6;
echo($sum);
?>
```

- Source 45

Logiske operatorer

operator	Beskrivelse
!	NOT
&&	AND
||	OR
and	
xor	
or	

```
<?php
if (file_exists(”navne.txt”) && is_readable(“navne.txt”)) {
    fopen(“navne.txt”,r);
    echo(”navne.txt er åben”);
} else {
    echo(”navne.txt er ikke åben”);
}
?>
```

- Source 46

Arrays

Til at gemme grupper af data enheder i en variabel, kan du bruge arrays. I PHP er der små forskelle til konventionelle programmeringssprog som C og pascal.

```
<?php
$sprog[] = ”Dansk”;
$sprog[] = ”Norsk”;
$sprog[] = ”svensk”;
echo($sprog[1]);
?>
```

- Source 47

Operator prioriteter

operator	Kommentar
`()`	
`new`	
`[]`	
`! ~ ++ -- + - (int) (double) (string) (array) (object) @`	+ og – som fortegn
`* / %`	
`+ - .`	
`<< >>`	

```
< <= > >=
== != === !==
&
^
|
&&
||
? :
= += -= *= /= .= &= |= ~= <<= >>=
print
and
xor
or
,
```

Variabler fra verdenen udenfor

PHP har konfiguration i filen php.ini som kan være afgørende for om du kan bruge variabler udenfor PHP.

System og GET variabler og $HTTP_Array

GET variabler er parametre til URL'en (Adresse).

```
http://www.firma.dk/test.php?frugt=banan&pris=2.25
```

Prøv med en adresse som ovenstående til dette program:

test.php

```
<?php
echo($frugt);
echo("<br>");
echo($pris);
?>
```

- Source 48

Formularer kan give variabler som GET variabler.

test.htm

```
<form action="test.php" method="get">
frugt: <input type="text" name="frugt">
pris: <input type="text" name="pris">
<input type="submit">
</form>
```

- Source 49

POST variabler

POST variabler bliver aldrig vist i adresselinien, men overføres fra browseren i headeren, som browseren ikke viser. Når man bruger POST variabler i en formular bruges method=”post” i stedet for method=”get”.

test.htm

```
<form action=”test.php” method=”post”>
frugt: <input type=”text” name=”frugt”>
pris: <input type=”text” name=”pris”>
<input type=”submit”>
</form>
```

- Source 50

Cookies

PHP kigger i headeren fra browseren efter en ”Cookie:” header i en request. Kort fortalt er det navne-værdier i par som kan sendes til browseren med ”Set-cookie:” i HTTP-headeren.

CGI variabler

PHP danner de standard CGI environment variabler som repræsenterer dele af informationer om den request der ledte til programmet som eksekveres. Disse inkluderer:

Variabel	Beskrivelse
$DOCUMENT_ROOT	Den lokale filsystems sti til det directory som indeholder scriptet.
$REMOTE_ADDR	IP adressen på den der laver request på siden.
$REMOTE_PORT	Portnummer på den der laver request på siden lytter.
$SCRIPT_FILENAME	Den lokale filsystems sti til PHP eksekverbare.
$SERVER_ADDR	IP adressen på den maskine som webserveren kører på.
$SERVER_NAME	Host navnet på den maskine som webserveren kører på
$SERVER_PORT	Port nummeret som webserveren lytter på.
$SERVER_PROTOCOL	HTTP versionen med hvilken klient og server kommunikerer.
$REQUEST_METHOD	HTTP metoden hvormed klienten laver request af siden (GET eller POST).
$QUERY_STRING	Den af URL’en som klienten lavede request med efter ?.
$REQUEST_URI	Den af URL’en som klienten lavede request med efter hostname og port.
$PHP_SELF	Den sti som klienten skal

HTTP header variabler

HTTP Header	Variabel	Beskrivelse
Host:	$HTTP_HOST	Navnet på den host som klienten tror den forbinder sig til. Det er ikke nødvendigvis den samme som $SERVER_NAME hvis webserveren har flere navne.
User-agent:	$HTTP_USER_AGENT	Klientens browser.
Accept:	$HTTP_ACCEPT	En liste af MIME typer som klientens browser kan acceptere.
Accept-language:	$HTTP_ACCEPT_LANGUAGE	En liste af 2tegns sprogkoder som klienten foretrækker.

PHP Strukturer

Program flow kontrolstrukturer

Betingede sætninger

if

```
if (condition) statement;
```

```
<?php
$bIsMorning = true;
if ($bIsMorning) {
    $sHilsen = “God morgen”;
    echo($sHilsen);
}
?>
```

- Source 51

```
<?php
$bIsMorning = true;
if ($bIsMorning) {
    $sHilsen = “God morgen”;
}
if (!$bIsMorning) {
    $sHilsen = “Hej”;
}
echo($sHilsen);
?>
```

- Source 52

```
<?php
$bIsMorning = true;
if ($bIsMorning) {
```

```
    $sHilsen = “God morgen”;
} else {
    $sHilsen = “Hej”;
}
echo($sHilsen);
?>
```

- Source 53

```
<?php
$bIsAfternoon = true;
if ($bIsMorning) {
    $sHilsen = “God morgen”;
} elseif ($bIsAfternoon) {
    $sHilsen = “God eftermidddag”;
}
} elseif ($bIsEvening) {
    $sHilsen = “God aften”;
} else {
    $sHilsen = “Hej”;
}
echo($sHilsen);
?>
```

- Source 54

```
<?php
$iHour = 13;
if ($iHour < 12) {
    $sHilsen = “God morgen”;
} elseif ($iHour < 17) {
    $sHilsen = “God eftermidddag”;
} else {
    $sHilsen = “God aften”;
}
echo($sHilsen);
?>
```

- Source 55

```
<?php
$sSaeson = ”sommer”;
if ($sSaeson == ”sommer”) {
    $fPris = 12.45;
} else {
    $fPris = 11.20;
}
?>
```

Samme som ovenstående med betingelsesoperator

```
<?php
$sSaeson = ”sommer”;
$fPris = ($sSaeson == ”sommer” ? 12.45 : 11.20);
?>
```

- Source 56

switch

```
switch (expression) {
case value1:
    statements;
    break;

case value2:
    statements;
    break;

default:
    statements;
}
```

```
<?php
$sSprog = ”da”;
switch ($sSprog) {
case ”fr”:
    echo ”Fransk”;
    break;
case ”de”:
    echo ”Tysk”;
    break;
case ”en”:
    echo ”Engelsk”;
    break;
case ”da”:
    echo ”Dansk”;
    break;
default:
    echo ”Ukendt sprog”;
}
?>
```

- Source 57

Almindelig fejl

```
<?php
$sSprog = ”fr”;
switch ($sSprog) {
case ”fr”:
    echo ”Fransk”;
// falder igennem
case ”de”:
    echo ”Tysk”;
    break;
case ”en”:
    echo ”Engelsk”;
    break;
case ”da”:
    echo ”Dansk”;
    break;
default:
```

```
    echo ”Ukendt sprog”;
}
?>
```

- Source 58

Løkker

while

```
while (condition) {
    statements
}
```

while looper fra 0 til flere gange

```
<?php
$iMax_Spillere = 8;
echo(”<select name=\”ant_spillere\”>\n”);
$i = 0;
while (++$i <= $iMax_Spillere) {
    echo(”<option value=\”$i\”>$i</option>\n”);
}
echo(“</select>”);
?>
```

- Source 59

break stopper loops

```
<?php
$iMax_Spillere = 20;
echo(”<select name=\”ant_spillere\”>\n”);
$i = 0;
while (++$i <= $iMax_Spillere) {
    if ($i > 10) {
        break; //stop
    }
    echo(”<option value=\”$i\”>$i</option>\n”);
}
echo(“</select>”);
?>
```

- Source 60

continue springer til næste loop

```
<?php
$iMax_Spillere = 20;
echo(”<select name=\”ant_spillere\”>\n”);
$i = 0;
while (++$i <= $iMax_Spillere) {
    if ($i % 2 == 0) {
        continue; //spring til næste loop
    }
```

```
    echo(”<option value=\”$i\”>$i</option>\n”);
}
echo(“</select>”);
?>
```

- Source 61

do ... while

```
do {
    statements
} while (condition)
```

do-while looper fra 1 til flere gange

```
<?php
$iMax_Spillere = 20;
echo(”<select name=\”ant_spillere\”>\n”);
$i = 0;
do {
    if ($i % 2 == 0) {
        continue; //spring til næste loop
    }
    echo(”<option value=\”$i\”>$i</option>\n”);
} while (++$i <= $iMax_Spillere);
echo(“</select>”);
?>
```

- Source 62

Både break og continue gælder for alle løkke konstruktioner.

for

```
for (expression1; expression2; expression3) {
    statements
}
```

```
<?php
$iMax_Spillere = 20;
echo(”<select name=\”ant_spillere\”>\n”);
$i = 0;
for ($i=0; $i <= $iMax_Spillere; ++$i) {
    if ($i % 2 == 0) {
        continue; //spring til næste loop
    }
    echo(”<option value=\”$i\”>$i</option>\n”);
}
echo(“</select>”);
?>
```

- Source 63

Function

Definer function

```
<?php
function dobbel($i)
{
    $i = 2 * $i;
    return $i;
}

$a = 8;
$n = dobbel($a);
echo($n);
echo("<br>");
echo($a);
?>
```

- Source 64

Adresse-operator & giver adgang til ændringer i parameter variabler udenfor funktionen.

```
<?php
function dobbel(&$i)
{
    $i = 2 * $i;
    return $i;
}

$a = 8;
$n = dobbel($a);
echo($n);
echo("<br>");
echo($a);
?>
```

- Source 65

```
<?php
function moms(&$fTot, $fMomspct)
{
    $fTot = $fTot * (1 + $fMomspct/100);
}

$fBeloeb = 100.0;
$fTotal = $fBeloeb;
moms($fTotal,25.0);
echo($fBeloeb);
echo("<br>");
echo($fTotal);
?>
```

- Source 66

Variabel scope

Variabler deklareret i en function har lokal scope. Ændringer i en functions variabler påvirker ikke variabler udenfor functionen selvom der findes en variabel udenfor functionen der har samme navn, de har hver deres memory placering. Hvad der er specielt for PHP er, at globale variabler ikke umiddelbart har noget scope indenfor functionen. Du skal bruge en prefix **global** for at få tilgang til globale variabler indenfor en function.

```
<?php
function udskrivbeloeb()
{
    $fBeloeb = 120.0;
    echo($fBeloeb);
}

$fBeloeb = 90.0;
echo($fBeloeb);
echo(”<br”>);
udskrivbeloeb();
echo(”<br”>);
echo($fBeloeb);
?>
```

- Source 67

```
<?php
function udskrivbeloeb()
{
    global $fBeloeb;

    $fBeloeb = 120.0;
    echo($fBeloeb);
}

$fBeloeb = 90.0;
echo($fBeloeb);
echo(”<br”>);
udskrivbeloeb();
echo(”<br”>);
echo($fBeloeb);
?>
```

- Source 68

```
<?php
function udskrivbeloeb()
{
    $GLOBALS[”fBeloeb”] = 120.0;
    echo($GLOBALS[”fBeloeb”]);
}

$fBeloeb = 90.0;
echo($fBeloeb);
echo(”<br”>);
udskrivbeloeb();
echo(”<br”>);
echo($fBeloeb);
```

```
?>
```

- Source 69

> Med mindre det er absolut nødvendigt, bør globale variabler undgås

Variabel levetid

Lokale variabler eksisterer kun så længe der er noget der refererer til den. PHP sørger for at fjerne variabler der ikke længere bruges gennem en ”garbage collector”.

```
<?php
function counter() {
    $i = 0;
    return ++$i;
}

echo(counter());
echo(“<br>”);
echo(counter());
?>
```

- Source 70

vil altid returnere 1

Hvis du vil kan du lave bruge prefix static til variablen og med denne bliver variablens værdi der stadig gennem alle kald. Dog bliver variablen initialiseret på ny når websiden reloades.

```
<?php
function counter() {
    static $i = 0;
    return ++$i;
}

echo(counter());
echo(“<br>”);
echo(counter());
?>
```

- Source 71

Rekursion

```
<?php
function produkt($m)
{
    if (m == 1) {
        $res = 1;
    } else {
        $res = $m * produkt($m – 1);
    }
    return $res;
}
```

```
echo(produkt(5));
?>
```

- Source 72

ovenstående skulle give en fejl fordi den kører igen og igen, ret fejlen og prøv igen.

Tildeling af function til variabler

```
<?php
function add($a,$b)
{
    return $a + $b;
}

function mul($a,$b)
{
    return $a * $b;
}

function RegneFunk($sOpr)
{
    switch ($sOpr) {
    case "+":
        $regnefunktion = "add";
        break;
    case "*":
        $regnefunktion = "mul";
        break;
    }
    return $regnefunktion;
}

$iA = 2;
$iB = 4;
$sOpr = "+";
$regnefunktion = RegneFunk($sOpr);
echo($regnefunktion($iA,$iB)."<BR>");

$sOpr = "*";
$regnefunktion = RegneFunk($sOpr);
echo($regnefunktion($iA,$iB)."<BR>");
?>
```

- Source 73

Brug af funktioner til organisering af kode

Du ser et skelet til organisering af kode. Fyld functioner med kode så du kan lace en test. Koden skal bruge en variabel $action som den skal have fra en parameter til url'el dvs. f.eks. fra en ekstern html-side hvor du laver en form. Som du ser i main function forventer action "","Create","Delete" eller "View".

```
<?php
function validateData()
{
    //Placer kode
```

```
}

function createRecord()
{
    //Placer kode
}

function deleteRecord()
{
    //Placer kode
}

function getData()
{
    //Placer kode
}

function displayMenu()
{
    //Placer kode
}

function displayResults($b)
{
    //Placer kode
}

function displayData($a)
{
    //Placer kode
}

function main()
{
    global $action;

    if (!validateData()) {
        //Placer kode
    }

    switch ($action) {
    case "":
        //Der blev ikke trykket på knappen
        //Vi siger det er første besøg på siden
        displayMenu();
        break

    case "Create":
        $bSuccess = CreateRecord();
        displayResult($bSuccess);
        break

    case "Delete":
        $bSuccess = DeleteRecord();
        displayResult($bSuccess);
        break

    case "View":
        $aData = getData();
```

```
        displayData($aData);
        break
    }
} // end main()

// Kald main()
main();

?>
```

- Source 74

Opgave: Du lavede en ekstern html-side med en formular. Den html-side skal du putte ind i php-koden under displayMenu() functionen.

Arrays

Et array er et element der holder flere værdier, hvert element udpeges med dets index (eller nøgle). Den mest almindelige type er numerisk integer som indicer. Numerisk baserede indicier er med nul base, første element har et index på 0.

Initaliser Arrays

```
<?php
$sprog[] = ”Dansk”;
$sprog[] = ”Norsk”;
$sprog[] = ”svensk”;
echo($sprog[1]);
?>
```

- Source 75

```
<?php
$sprog[0] = ”Dansk”;
$sprog[1] = ”Norsk”;
$sprog[2] = ”svensk”;
echo($sprog[1]);
?>
```

- Source 76

```
<?php
$sprog[100] = ”Dansk”;
$sprog[300] = ”Svensk”;
$sprog[200] = ”Norsk”;
$sprog[] = ”Engelsk”;

echo($sprog[200]);
echo(”<BR>”);
echo($sprog[301]);
?>
```

- Source 77

```
<?php
$sprog = array(”Dansk”, ”Svensk”, ”Norsk”, ”Engelsk”);

for ($i=0; $i<4; $i++) {
    echo($i.”: ”.$sprog[$i]); echo(”<BR>”);
```

```
}
?>
```

- Source 78

```
<?php
$sprog = array(”Dansk”, 3 => ”Svensk”, ”Norsk”, ”Engelsk”);

for ($i=0; $i<6; $i++) {
    echo($i.”: ”.$sprog[$i]); echo(”<BR>”);
}
?>
```

- Source 79

```
<?php
$sprog = array(
    ”da” => ”Dansk”,
    ”se” => ”Svensk”,
    ”no” => ”Norsk”,
    ”en” => ”Engelsk”);

echo($sprog[”se”]);
?>
```

- Source 80

Loop igennem et array

```
foreach (array as [$key =>] $value) {
    statements
}
```

```
<?php
$sprog = array(
    ”da” => ”Dansk”,
    ”se” => ”Svensk”,
    ”no” => ”Norsk”,
    ”en” => ”Engelsk”);

foreach ($sSprog as $sIdx => $sVal) {
    echo($sIdx.”: ”.$sVal); echo(”<BR>”);
}
?>
```

- Source 81

```
<?php
$sprog = array(
    ”da” => ”Dansk”,
    ”se” => ”Svensk”,
    ”no” => ”Norsk”,
    ”en” => ”Engelsk”);
echo(”Sprog er:<br><ul>\n”);
foreach ($sSprog as $sVal) {
    echo(”<li>”.$sVal.”</li>\n”);
}
echo(”</ul>\n”);
```

```
?>
```

- Source 82

Indbyggede Array functions

count()

```
int count(mixed var)
```

```
<?php
$a1 = array(”aaa”,”bbb”,”ccc”);
$a2 = array(”aaa”,”bbb”,”ccc”,”ddd”);
echo(”Antal elementer i \$a1:”.count($a1).”<br>\n”);
echo(”Antal elementer i \$a2:”.count($a2).”<br>\n”);
?>
```

- Source 83

in_array()

```
boolean in_array(mixed needle, array haystack [, bool
   strict])
```

```
<?php
$a = array(”aaa”,”bbb”,”ccc”,”ddd”);
echo(in_array(“ccc”,$a).”<br>\n”);
?>
```

- Source 84

reset()

```
mixed reset(array array)
```

Ethvert array i PHP har en intern peger som holder styr på aktuel position i et arrat. Når du bruger konstruktioner som foreach foretages en reset på pegeren til begyndelsen af array’et.. Mange array-function’er, som f.eks next() og prev() flytter pegeren til en ny position i array’et- Dette kan påvirke fremtidige kald til function’er som array_walk som begynder på det aktuelle sted som pegeren befinder sig.

array_walk()

```
<?php
function xxx($sInd)
{
    echo($sInd.”<BR>”);
}

$a = array(“aaa”,”bbb”,”ccc”);
```

```
reset($a); //stil peger til første element i array
array_walk($a,"xxx");

foreach ($a as $sIdx => $sVal) {
    echo($sIdx.": ".$sVal); echo("<BR>");
}
?>
```

- Source 85

sort()

```
void sort(array array [, int sortflags])
```

Hvor sortflags er SORT_REGULAR, SORT_NUMERIC eller SORT_STRING.

```
<?php
$sprog = array(
    "da" => "Dansk",
    "se" => "Svensk",
    "no" => "Norsk",
    "en" => "Engelsk");

echo("Sprog (usorteret) er:<br><ul>\n");
foreach ($sSprog as $sVal) {
    echo("<li>".$sVal."</li>\n");
}
echo("</ul>\n");
sort($sprog)

echo("Sprog (sorteret) er:<br><ul>\n");
foreach ($sSprog as $sVal) {
    echo("<li>".$sVal."</li>\n");
}
echo("</ul>\n");
?>
```

- Source 86

Se også online dokumentation for `arsort(), asort(), ksort(), natsort(), natcasesort(), rsort(), usort(), array_multisort()` og `uksort()`.

explode() og implode()

Disse to function'er betragtes som hørende til string-function kategorien, men de har også med array's at gøre.

```
<?php
$a = array("aaa","bbb","ccc");
$s = implode('; ',$a);
echo($s."<BR>");

$s = "Dansk Engelsk Tysk Fransk";
$a = explode(' ',$s);
foreach ($a as $sIdx => $sVal) {
    echo($sIdx.": ".$sVal); echo("<BR>");
}
```

```
?>
```

- Source 87

Predefinerede arrays

Du har i PHP adskillige variabler som gemmes i array's. $GLOBALS, $HTTP_GET_VARS, $HTTP_POST_VARS og $HTTP_COOKIE_VARS.

Flerdimensionelle array's

```
<?php
$aSprog= array(
  "slavisk" => array("rusisk","polsk","Slovakisk"),
  "Germansk" => array("Svensk", "Hollandsk", "Engelsk"),
  "Romansk" => array("Italiensk", "Spansk", "Rumænsk")
);
foreach ($aSprog as $sIdx => $aFamilie) {
  echo(
   "<h2>$sIdx</h2>\n" .
   "<ul>\n" //start listen
  );
  foreach ($aFamilie as $sSprog) {
    echo("<li>$sSprog</li>\n");
  }
  echo("</ul>\n");
}
?>
```

- Source 88

Objekt-Orienteret Programmering med PHP

Class

```
class ClassName [extends parentClassName]
{
   var $member1;
   var $member2;
   ...
   var $memberN;

   //Constructor
   function ClassName()
   {
   }

   function method1()
   {
   }
```

```
   function method2()
   {
   }

...

   function methodN()
   {
   }
}
```

- Source 89

Betragt følgende medietype:

<u>Media</u>
id
name
type
inStock
price
rating

<u>cd</u>
serialNo
artist
numberOfTracks
trackNames

<u>software</u>
serialNo
publisher
platform
requierements

<u>Movie</u>
serialNo
minutes
director
cast
mediatype (dvd/vhs)

<u>Book</u>
isbn
author
numberOfPage

Disse vil vi bruge som et eksempel på OOP-programmering, se kildetekster preoop.php, media.php, book.php og oop.php.

Bruger input og Regular expressions

Regular expression

Regular expressions (**regex**) conceptet blev introduceret af Stephen Kleene i 1956 til en matematisk beskrivelse af neuro-psykologiske fænomener. Ken Thompson, den oprindelige opfinder af Unix gjorde et arbejde i frembringningen af den første applikation der brugte denne algebra.

Regular Expression i PHP

Regular expression er et meget brugt udtryk sammen med Unix. Det er en metode til at søge/erstatte tekster som bruges i flere programmeringssprog til Unix og den er standardiseret som **POSIX resular expression**. Der findes andre standarder som divergerer fra denne, men til PHP holder POSIX standarden. Hvis du interesseret i mere om standarden se. http://www.pasc.org.

I php findes funktioner ereg(), ereg_replace(), eregi(), eregi_replace().

ereg()

```
int ereg(string pattern, string string [, array regs])
```

```
<?php
$s = "abcxyzabc";
echo ereg("xyz",$s) ? "sand" : "falsk";
?>
```

- Source 90

ereg_replace()

```
string ereg_replace(string pattern, string replacement,
   string string)
```

```
<?php
$s = "Der rettes skal i dette";
echo ereg_replace("( )rettes( )skal",
   "\\1skal\\2rettes",$s);
echo "<br>";
echo ereg_replace("(rettes) (skal)","\\2 \\1",$s);
?>
```

- Source 91

Når pattern indeholder substrings (paranteser) kan \\tal bruges til erstatninger. \\0 repræsenterer hele strengen og \\1 første substring optil 9 substring kan anvendes.

eregi()

```
int eregi(string pattern, string string [, array regs])
```

Samme funktion som ereg(), men uden forskel på store og små bogstaver.

eregi_replace()

```
string eregi_replace(string pattern, string
   replacement, string string)
```

Samme funktion som ereg_replace(), men uden forskel på store og små bogstaver.

split()

```
array split(string pattern, string string [,int limit])
```

```
<?php
$sDato = “24/12-2003 er det juleaften igen”;
$aDato = split(’[/. -]’,$sDato,4);
foreach ($aDato as $s) {
    echo “$s <br>”;
}
?>
```

- Source 92

spliti()

```
array spliti(string pattern, string string [,int
   limit])
```

sql_regcase()

```
string sql_regcase(string string)
```

Syntaks basis

Dette udtryk vil matche en vilkårlig streng som indeholder “xyz”:

```
“xyz”
```

Lav nedenstående test program. Den inddeholder en funktion test_ereg() som laver et format for resultater.

```
<?php
function test_ereg($pattern,$a)
{
    echo "pattern \"$pattern\" matcher ";
    echo "<table border=1>";
    echo "<tr>";
    foreach ($a as $s) {
        echo "<td align=center>\"$s\"</td>";
    }
```

```
    echo "</tr><tr>";
    foreach ($a as $s) {
        echo "<td align=center>";
        $r = ereg($pattern,$s,$aReg);
        echo( $r ? "ja" : "nej");
        if ($r) {
            foreach ($aReg as $sReg) {
                if ($sReg) {
                    echo "($sReg)";
                }
            }
        }
        echo "</td>";
    }
    echo "</tr>";
    echo "</table>";
}

// lav nogle teststrenge i $a
$a = array('abcxmmm','abc','xyz'
          ,'abcxyz','xyzabc','mmmxyzmmm','mmmyzmmm');
// test $a strengene med patter 'abc'
test_ereg('abc',$a);
?>
```

- Source 93

Enhver streng som indeholder enten "abc" eller "xyz" vil matche dette udtryk:

```
"abc|xyz"
```

```
<?php

$a = array('abcxmmm','abc','xyz'
          ,'abcxyz','xyzabc','mmmxyzmmm','mmmyzmmm');
test_ereg('abc|xyz',$a);
?>
```

- Source 94

Bracket expression.

Dette vil matche en streng der indeholder mindst et af bogstaverne "x", "y" eller "z":

```
"[xyz]"
```

Matcher kun tal:

```
"[0123456789]"
```

en kortere måde at skrive det på er:

```
"[0-9]"
```

Exclude bracket expression.

Match alle tegn der IKKE er ”x”, ”y” eller ”z”:

```
“[^xyz]”
```

```
<?php

$a = array('abcxmmm','abc','xyz'
          ,'abcxyz','xyzabc','mmmxyzmmm','mmmyzmmm');
// bracket expression
test_ereg('[xyz]',$a);
// exclude
test_ereg('[^xyz]',$a);
$a = array('123','asd','a9s8');
test_ereg('[0123456789]',$a);
test_ereg('[0-9]',$a);
test_ereg('[^0-9]',$a);
?>
```

- Source 95

Qualifiers

Qualifiers ”+”, ”*” og ”?” gentager et match:

- ”x+” matcher 1 eller flere af bogstavet ”x”.
- ”x*” matcher 0 eller flere af bogstavet ”x”.
- ”x?” matcher 0 eller 1 af bogstavet ”x”.

Bounds

Bounds er tal i tuborg-paranteser. I lighed med qualifiers gentages en søgning, men du kan afgrænse antallet vilkårligt.

- ”ab{3}” matcher et ”a” efterfulgt af nøjagtigt 3 ”b”er.
- ab{3,} mindst 3 ”b”er.
- ab{3,5} mellem 3 og 5 ”b”er.

```
<?php

$a = array('123','abc','abxc','abxxc','abxxxc','abxxxxc');
// qualifiers
test_ereg('bx+c',$a);
test_ereg('bx*c',$a);
test_ereg('bx?c',$a);
$a = array('abc','abbc','abbbc'
          ,'abbbbc','abbbbbc','abbbbbbc','abbbbbbbc');
// bounds
test_ereg('ab{3}c',$a);
```

```
test_ereg('ab{3,}c',$a);
test_ereg('ab{3,5}c',$a);
?>
```

- Source 96

Paranteser

For at sammenstykke sekvenser af tegn putter du dem i paranteser disse udtryk kan kombineres med qualifiers og bounds.

- x(yz)* matcher et ”x” efterfulgt af 0 til flere forekomster af ”yz”
- x(yz({3,5} matcher x efterfulgt af 3 til 5 forekomster af ”yz”

```
<?php

$a = array('abc','abxyc','abxyxyc'
          ,'abxyxyxyc','abxyxyxyxyc'
          ,'abxyxyxyxyxyc','abxyxyxyxyxyxyc'
          ,'abxyxyxyxyxyxyxyc');
// paranthesis
test_ereg('ab(xy)*c',$a);
test_ereg('ab(xy){3,5}c',$a);
?>
```

- Source 97

Special tegn

- ”.” matcher et vilkårligt tegn.
- ”^” matcher starten af en streng.
- ”$” matcher enden af en streng.
- ”[[:alnum:]]” matcher alle alfanumeriske tegn svarer til ”[a-zA-Z0-9]”.
- ”[[:digit:]]” matcher alle tal svarende til ”[0-9]”.
- ”[[:alpha:]]” matcher alle bogstaver svarer til ”[a-zA-Z]”.

```
<?php

$a = array('abc','abxyc','abxyxyc'
          ,'abxyxyxyc','abxyxyxyxyc'
          ,'abxyxyxyxyxyc','abxyxyxyxyxyxyc'
          ,'abxyxyxyxyxyxyxyc');

// special characters “.”, “^” og “$”
test_ereg('ab.c',$a);
test_ereg('ab..c',$a);
test_ereg('ab.+c',$a);
test_ereg('ab.{4,6}c',$a);
$a = array('abc','abcx','xabc','xabcx');
```

```
test_ereg('^ab',$a);
test_ereg('bc$',$a);
// character classes
$a = array('123','abc','ab8c',',+=','<>','abxxxxc','æøå');
test_ereg('[[:alnum:]]',$a);
test_ereg('[^[:alnum:]]',$a);
test_ereg('[[:digit:]]',$a);
test_ereg('[^[:digit:]]',$a);
test_ereg('[[:alpha:]]',$a);
test_ereg('[^[:alpha:]]',$a);
?>
```

- Source 98

Sammensætning af regular expression.

Lad os lave et udtryk der kan validere et beløb. Hvis vi kigger på nogle valide eksempler:

0
10000
10.000
10000,50
10.000,50

kan vi opstille kriterier. Beløb skal tilfredsstille følgende kriterier:

- Det skal være enten 0 eller et beløb der ikke stater med 0 med mindre det er det eneste ciffer foran et decimalkomma.
- Det kan have optil 2 cifre efter kommaet.
- Det kan være negativt altså med foranstillet ”-”.
- Det kan have punktum som tusindseperator.

```
'^0$' //0

'(\,[0-9]{1,2})?$' //,00

'^[1-9][0-9]*$'  //1000000

'^-?[1-9][0-9]*$' //-10000000

'(\.[0-9]{3})*$'  // .000.123....

'^0|-?[1-9][0-9]*$' // 0 og -10000000

'^(0|-?[1-9][0-9]*)(\,[0-9]{1,2})?$' // 0,50 og –1000,50

'^(-?0|-?[1-9][0-9]*(\.[0-9]{3})*)(\,[0-9]{1,2})?$'

'^-?(0|[1-9][0-9]*(\.[0-9]{3})*)(\,[0-9]{1,2})?$'
```

Perl Compatible Regular Expressions

Fra version 3.0.9 har PHP implementeret et udvalg af Perl-compatible regular expression (PCRE) funktioner. se. http://www.pcre.org.

Modifier	Beskrivelse
i	Case insensitive
x	Ignorer white space i pattern.
e	Kun brugt i preg_replace(). Denne funktion laver normal substituering af \\ referencer i erstatningsstrengen.

Shortcut	Beskrivelse
\d	Match decimaltal.
\D	Match ikke decimaltal.
\s	Match whitespace.
\S	Match ikke whitespace.
\w	Match “ord”.
\W	Match ikke “ord”

Shortcut	Beskrivelse
\b	Ord grænse.
\B	Ikke ord grænse.
\A	Start på subject (uafhængig af multiline mode)
\Z	Slut på subject eller ny linie i enden (uafhængig af multiline mode)
\z	Slut på subject (uafhængig af multiline mode)

Skabelon

•

Parameter	Beskrivelse

Kode
<?php ?>
Output

program

1.	
2.	

Personer

Navn	Adresse	Postby
Anders Nielsen	Kirkegade 14	7430 Ikast
Bodil Pedersen	Søndergade 19	7430 Ikast