Malmö högskola
Centrum för teknikstudier

Webbprogrammering, DA123A, Studiemateriel 8

Flerstegsformulär

Ett flerstegsformulär är ett stort formulär uppdelat på flera sidor där data hämtas från varje sida och påverkar utseendet på nästa sida som genereras.
Fördelar:

- Lättare för besökaren att greppa frågorna och mata in riktiga uppgifter.
- Inte så stora sidor med många frågor som skrämmer iväg användare.

Nackdelar:

- HTTP är statuslöst. Vi måste på något vis komma ihåg redan postad data mellan gångerna, och detta sker ej med automatik.

Eftersom HTTP är statuslöst måste vi, efter att till exempel hämtat information från en sida och skicka den till nästa sida, ha metoder för detta. Det finns fyra vanliga metoder för att skicka data mellan sidorna via formulär.

- *Formulärvariabler* - (kallas ibland ”gömda variabler" eller hidden) Dessa är inte en direkt del utav PHP utan är kopplade till metoderna GET och POST.
- *Sessions-variabler* - Valda data sparas på servern och på din dator, i en separat fil respektive en cookie-fil (PHPSESSID) och en unik indentifikationssträng skickas till användaren när en session börjar.
- *Cookies* - (fungerar även utan PHP) Kan också användas för att mellanlagra information på användarens hårddisk.
- *Globala variabler* – variabler som skapas utifrån formuläret och som lagras mellan sidorna. Medför dock vissa säkerhetsrisker som nämnts tidigare.

Cookies

Cookien(eller smulan , se sm 6) sänds som en del i header-informationen i filen. Precis som i JavaScript har smulan olika argument som måste sättas:

- name: namnet som används för att referera till smulan hos kakan.
- value: värdet hos smulan.
- expiration: sätts den ej så kommer cookien att bli en sessions-cookie och raderas efter man stängt webbläsaren.

path, domain, security är frivilliga uppgifter.

När det gäller att sätta cookies/smulor så måste detta göras innan någon output sker till webbläsaren. Detta innebär i praktiken att om du skall sätta en smula med PHP på en sida så måste du göra detta i ett stycke php-skript som finns först i filen. Det får inte ens finnas en tom rad innan första <?php-taggen då denna rad i sådana fall går som output till webbläsaren.

Om man vill strukturera sin sida så att det sker output i koden eller mellan kodblock innan man sätter en cookie/smula så kan man använda sig av bufferfunktioner för att förhindra detta. Funktionen *ob_start()* anger att all output skall lagras tills vidare. Om ingen output skett innan ob-start anropas så kan du fortfarande skicka headers och då bland annat sätta en cookie/smula. När skriptet har utfört det som behövs innan output genereras till klienten så anropar man funktionen ob_end_flush() för att skicka den output som finns i bufferten och stänga av bufferlagringen.

Fördelen med att använda cookies med PHP jämfört med JavaScript är att du läser cookien/smulan innan du skickar outputen till klienten. Du kan anpassa innehållet efter cookien/smulan innan sidan når klienten och inget extra arbete sker på klientsidan.

Kod som sätter en cookie eller snarare en smula:

```
<?php
$cookie_name = "test";
$cookie_value = "Sten Sture den äldre";
$cookie_expire = time()+86400;
setcookie($cookie_name, $cookie_value, $cookie_expire);
?>
```

Kod som använder den satta smulan på en annan sida:

```
<?php
if(isset($_COOKIE['test']))
  echo "Välkommen " . $_COOKIE['test']. "!<br>";
else
  echo "Det är fel <br>";
?>
```

Utskriften på den andra sidan(om den första körts korrekt) blir nu:
Välkommen Sten Sture den äldre!

Observera att du inte behöver hämta hela cookien och sedan själv hitta den smula du söker utan att du direkt hittar smulan genom att ange dess namn tillsammans med den inbyggda variabeln $_COOKIE som fungerar som en array.

Ett exempel med bufferlagring:

```
<?php
ob_start();
echo "Hello World";
setcookie("cookie_name", "cookie_value");
ob_end_flush();
?>
```

Sessionsvariabler

Sessionsvariabler är variabler som är knutna till en användare - en session med ett sessions-ID. Sessionsvariabler finns på servern och kan på så sätt användas för att föra över information mellan sidor. Dessa kan exempelvis användas för att:

- hålla reda på information om inloggad användare
- flerstegs-formulär
- automatisk inloggning
- forum

Det finns två sätt att förmedla sessions-ID mellan sidor – cookies och direkt i URL:en. Att använda cookies är att föredra men är inte alltid möjligt då användaren kanske inte tillåter cookies. De flesta tillåter dock cookies idag och vi kommer att fokusera på att använda cookies.

Den bakomliggande tekniken som får det att fungera med cookies innebär att servern sätter en cookie hos klienten med automatisk genererat session ID för att kunna identifiera klienten. Det skapas också en fil på servern som kan identifieras med hjälp av sessions-ID. Servern sparar användarens sessionsvariabler i $_SESSION[]-listan som lagras i filen på servern. Vid

nästa anrop hämtas session ID från cookien hos servern och sedan kan PHP-scriptet lätt återfå sina variabler ur session-listan i motsvarande fil.

session_start()
Är anropet som används för att registrera sessionen det vill säga aktivera automatiken. Detta anrop måste alltid kallas innan du använder sessionsvariabler. Session_start läser klientens session-id från klientens cookie eller sätter en ny cookie och fil om ingen tidigare session fanns.

Lagra och läsa i sessionsvariabel:
Börja skriptet med

```
session_start();
```

Att spara ned ett värde ”värde” i sessionsvariabeln ”namn”.

```
$_SESSION[namn] = "värde";
```

Värdena kan läsas från $_SESSION[namn].

```
$namn = $_SESSION[namn];
```

Import request variables

Om register_globals är avslaget och man vill ha in variabler sända via GET/POST/Cookie som globala, kan man använda funktionen import_request_variables().

import_request_variables (string types [, string prefix])

Funktionen ger variabler skickade via POST, GET och Cookie ett valfritt prefix. Använd prefixet, annars är det samma sak som om register globals hade varit påslaget – någon kan gissa sig till dina variabelnamn och på så sätt komma åt data som de inte borde. Denna metod kan vara praktisk om man har många funktioner som behöver använda samma variabler, men det ska verkligen vara en fördel mot att använda sig av exempelvis $_POST för att användandet av metoden skall vara befogat.

```
// vår URL: http://xxx/x.php?namn=olle&alder=15&nick=xxx
import_request_variables('g','formdata_');
```

Detta hämtar $formdata_namn, $formdata_alder och $formdata_nick som är skickade med GET-metoden. Bokstaven ”g” refererar till GET det vill säga det sättet variabeln i exemplet ovan är skickad på. Man kan även använda P och C för respektive Post och Cookie. Man kan även använda kombinationer av dessa. Då har den sista bokstaven högst prioritet, det vill säga om man anger ”pg” kommer get-variabler att skriva över post-variabler om dessa har samma namn.

Att definiera egna funktioner

Att definiera sina egna funktioner och kalla på dem är enkelt. Att kunna definiera sina egna funktioner är nödvändigt för att kunna dela upp sin kod i hanterbara delar. Indelning av kod i funktioner ökar läsbarheten. Det gör också att ofta förekommande kod kan utgöras av en funktion som anropas flera gånger istället för att ha samma kodsnutt på flera ställen i koden.

Exempel:

```
<?php
function skrivhej() {
 echo "<h2>hej</h2>"; //detta utförs varje gång funktionen anropas
}
skrivhej(); //anrop till funktionen skrivhej()
skrivhej(); //anrop till funktionen skrivhej()
//vi har nu skrivit hej som en heading av storlek 2, två gånger
?>
```

Endast fantasin sätter gränser på vad man kan göra/skapa med sina egna funktioner. När du programmerar så är det bra att vara uppmärksam på kod som förekommer ofta och undersöka om denna kod går att paketera i en funktion. Ofta kan man redan i förberedelserna för sin kod se att vissa moment upprepas ofta på mycket likartade sätt. Sådana saker är bra att skriva som egna funktioner.

Att skicka värden till funktioner.

En funktion kan ta emot indata (argument) för att till exempel utföra beräkningar på dessa. För att skicka data till funktioner som vi visat på innan används argument i funktionens "huvud". Exempel:

```
<?php
function testfunktion ($inargument1, $inargument2){
  echo ("<H$inargument1>".$inargument2."</H$inargument1>");
}
testfunktion (1, 'C-vitaminer');
testfunktion (2, 'är');
testfunktion (3, 'bra att äta');?>
```

Funktionen ovan tar inargument1 och inargument2 vilket i funktionen definieras som rubrikstorlek och textsträng.

Om vi till exempel vill ha en standardstorlek (default) som vi annars ska använda om det inte kommer någon inparameter för rubrikstorlek så kan vi ange detta på följande sätt.

```
<?php
function testfunktion ($inargument1, $inargument2=1){
 echo ("<h$inargument2>".$inargument1."</h$inargument2>");
}
testfunktion ('C-vitaminer',2);
testfunktion (är,4);
testfunktion ('bra att äta');
?>
```

Utskriften på skärmen om användaren låter bli att ge indata om storlek, den andra parametern, blir att texten formateras i rubrikstorlek1, <H1>. I ett verkligt praktiskt exempel skulle denna defaulttext vara formaterad till brödtext, till exempel 12pt.

Vi kan dock inte använda så kallade defaultvärden i det första argumentet men inte i det andra då skriptet då ej vet hur det ska tolka informationen som skickas till det i sådana fall. Defaultvärden (standardvärden) måste alltid komma sist i parameterdeklarationen.

Argument kommer att matchas med en ordning de kommer i anropet. Antag att vi har en funktion med totalt fem argument och de två första saknar deafultvärden och de tre sista har defaultvärden. Om vi anropar denna angivet tre argument så kommer de två sista argumenten att tolkas som att de skall ha defaultvärden men argument 1-3 kommer att ha de värden vi

anger i anropet. Så det går inte att ange 3 argument och mena att 1,2 och 4 ska ha de angivna värdena och 3 och 5 defaultvärden.

I en tabell så förekommer ju flera rader som ha mycket likartat utseende. Om vi skall generera en tabell i PHP så kan vi ju eka ut varje rad i tabellen för sig men det ger mycket mer kompakt och överskådlig kod om vi använder oss av en egendefinierad funktion och en for-sats:

```
<table border="1">
<?php
/* Funktion som skapar en tabellrad med två celler.
   Innehållet i den första cellen blir vad som finns i
   Parametern $kolumn1 och motsvarande för den andra cellen. */
function skapatabellrad( $kolumn1, $kolumn2) {
 print "<tr><td>$kolumn1</td><td>$kolumn2</td></tr>";
}
for ( $i=1; $i<=4; $i++ ) {
  $textvansterkol="rad $i kolumn 1";
  $texthogerkol="rad $i kolumn 2";
  skapatabellrad($textvansterkol, $texthogerkol);
}
?>
</table>
```

Resultat:

rad 1 kolumn 1	rad 1 kolumn 2
rad 2 kolumn 1	rad 2 kolumn 2
rad 3 kolumn 1	rad 3 kolumn 2
rad 4 kolumn 1	rad 4 kolumn 2

Ett annat exempel på en egendefinierad funktion

```
<?php
function summa($nummer1, $nummer2) {
  //Här kan det vara bra att deklarera vad funktionen gör
  //därför skriver vi här: Funktionen returnerar summan
  //av två tal. Inparametrarnas funktion, kan också beskrivas här.
  $temp = ($nummer1+$nummer2);
  return ($temp);
}
echo (summa(1,3));
?>
```

Ger utskriften 4.

Man kan också tilldela värdet av funktionen till en annan variabel genom att returnera ett värde från funktionen:

```
$tal=summa(1,3);
```

Med return stoppar man exekveringen av funktionen och tilldelar funktionen ett värde.

Man kan även anropa andra funktioner från en funktion.

Dynamiska funktionsanrop

Man kan tilldela funktioner till variabler genom att tilldela variabeln funktionens namn och därefter anropa funktionen med variabelns namn följt av parenteser. Detta kallas ett dynamiskt anrop. Exempel:

```
<?php
function helloworld () {
  echo "hello world"<br>";
}
$funktions_variabel = "helloworld";//observera utelämnandet av parenteser
$funktions_variabel();
?>
```

Dynamiska anrop används när man till exempel vill att ett program ska ändra sitt beteende på grund av en parameter i URL:n. Man kan då ta ut denna parameter och använda den till att starta (trigga) olika funktioner.

Funktioner och variabler

Globala och lokala variabler är variabler som nås av alla respektive vissa satser i till exempel en funktion. Exempel: I en sats utanför funktionen definieras variabeln $tal, men denna nås ej inne i funktionen.

```
<?php
$tal=11;
function test() {
 echo("Tal är: $tal”);
}
?>
```

Detta kan resultera i tre saker beroende lite på hur servern är konfigurerad och var man använder den odefinierade variabeln:

- Utskriften blir ”Tal är:” om $tal behandlas som en odefinierad variabel.
- Ett fel uppstår som avbryter tolkningen på denna rad och inget sker. Ibland så avbryts hela php-tolkningen.
- Ett felmeddelande genereras och skickas till webbläsaren

Vi har ju inte givit funktionen någon inparameter samt definierat $tal utanför funktionen. I funktionen finns alltså inte någon variabel med namnet $tal egentligen.

Annat exempel: I en sats inne i funktionen definieras variabeln $tal, men nås ej utanför funktionen.

```
<?php
function test() {
  $tal=11;
}
echo ("Tal är: $tal");
?>
```

Eftersom vi ej givit funktionen någon inparameter samt definierat $tal inne i funktionen så behandlas $tal utanför funktionen oftast som en ny variabel utan värde men något av punkterna ovan kan också ske.

Hur gör man då? Exempel: I en sats utanför funktionen definieras variablen $tal, och nås inne i funktionen på grund av att variabeln läses in i den globala variabeln $tal med hjälp av modifieraren global.

```
<?php
function test() {
global $tal;
$tal=11;
}
echo ("Tal är: $tal");
?>
```

Utskriften blir:
Tal är: 11

Detta fungerar på grund av att vi inne i funktionen definierar $tal som global variabel. Variabeln $tal nås alltså av alla funktioner och i hela programmet. Denna metod ska användas sparsamt då globala variabler i större program och sammanhang minskar läsbarheten och överskådligheten, samt kan ge upphov till komplikationer om flera ska editera koden.

Lägg också märke till skillnaden mellan PHP och de flesta andra programmeringsspråk: I de flesta språk så deklarerar man variabler som globala utanför funktioner för att sedan kunna använda dessa i funktioner. I PHP deklarerar man variabler som globala i funktioner för att kunna använda dem utanför funktioner.

Att tillåta globala variabler utgör en viss säkerhetsrisk som nämnt tidigare i samband med egenskapen register_globals hos servern och därför så kan man stänga av detta på servern. Så är fallet med denna kurs server dvwebb.mah.se. Därför så kan du inte testa exemplen ovan där. Att använda globala variabler leder oftast bara till svårläst kod och det finns egentligen ingen nytta av dem utan man kan alltid kan lösa detta med parametrar(eventuellt skickade som referenser) och returvärden. Mer om detta kommer längre fram.

Funktionstilldelning
För att tilldela en funktion ett värde till exempel efter en beräkning eller dylikt så använder man kommandot return. Exempel:

```
<?php
function addition($invar1, $invar2) {
  $resultat= $invar1+$invar2;
  return $resultat;
}
?>
```

Operationen kan göras direkt i return-satsen, enligt nedan

```
<?php
function addition($invar1, $invar2) {
  return ($invar1+$invar2);
}?>
```

Man kan också tilldela return-värdet av resultatet av andra funktioner. Exempel:

```
return (enannanfunktionsnamn($argumentnamn));
```

Att skicka värden från funktioner utan globala variabler.
Variabler som deklareras i en funktion beskrivs ofta som att de lever där och sedan dör när funktionen avslutas. Det finns olika metoder för att föra värden vidare som dessa variabler bär på.

Modifieraren static kan användas till att få en funktion att "hålla vissa av dess variabler vid liv" till nästa gång funktionen körs. Detta kan till exempel användas vid enkel uppräkning.

Exempel:

```
<?php
function upprakning(){
  static $iterationer=0;
  $iterationer++;
  echo("Ni har kört denna uppräkningen $iterationer gång(er)<BR>");
}
upprakning();
upprakning();
upprakning();
?>
```

Utskrift:

```
Ni har kört denna uppräkningen 1 gång(er)
Ni har kört denna uppräkningen 2 gång(er)
Ni har kört denna uppräkningen 3 gång(er)
```

Funktionen i exemplet ovan kommer alltså ihåg hur många gånger (iterationer) som den har exekverats. Detta sker på grund av att man deklarerar variabeln iterationer som static. Variabeln överlever alltså till nästa gång funktionen körs utan att för det vara global. Notera att vi inte skrivit någon return-sats i funktionen.

När man normalt skickar variabler till en funktion så tas de emot och behandlas i funktionen. Skickas de inte ut med return via funktionens namn eller är globala, så förblir de likadana som när de kom in sett utifrån programmet. Ingenting har hänt med variablerna vi skickade in till funktionen. Detta beror på att normalt skickar vi bara en kopia av värdet hos variabeln till funktionen, och det blir alltså en helt egen variabel i funktionen.

Ett annat sätt är att skicka argument via referens. Exempel:

```
<?php
function increment2(&$invarde){//observera &-tecknet framför variabeln
  $invarde +=2;
}
$startnummer=4;
increment2($startnummer);
echo($startnummer);?>
```

Resultat: 6

Om vi istället som i exemplet ovan skickar argumentet via referens (det vill säga vi skickar pekaren som pekar på positionen i minnet där variabeln lagras), genom att skriva ett &-tecken före parametern i funktionshuvudet när vi deklarerar increment2, så ändras värdet i variabeln av funktionen och variabeln behåller värdet efter det att programmet gått ur funktionen. Detta kan användas till att skicka ut flertalet variabler ur en funktion vilken kan vara mycket användbart. Med return-satsen kan vi ju endast skicka ut ett värde från funktionen.

Inbyggda funktioner

Inbyggda funktioner är funktioner som redan finns definierade i PHP och är av typen nytto-funktioner som underlättar saker som man ofta behöver göra.

function_exist() - Man kan kontrollera om funktioner existerar innan man använder dem, med den booleska funktionen function_exist(); Detta är nyttigt om man inte riktigt vet vilka versioner av PHP som ett skript kommer att köras på eller om man av någon orsak inte vet

vilka andra skript som är inkluderade. Genom att kontrollera existensen av en tveksam funktion så kan man undvika fel genom att då kunna förbereda ett alternativ.

is_numeric() - är en funktion som kontrollerar om en variabel är numerisk eller en sträng. Detta kan vara bra för att kontrollera indata.

rand() - är en funktion som slumpar tal. Exempel:

```
<?
if (function_exists("rand")){
  $tarning = rand(1,6);
  echo "Din tärning stannade på $tarning";
}
else echo "Nope, kan inte slumpa saker här inte";
?>
```

Tidigare behövde man ett trick som hette "seed" eller "ett frö" till sin randomfunktion i en extra funktion srand() för att den skulle generera slumptal med olika startvärde men efter PHP 4.2.0 så görs det automatiskt.

echo() - är en funktion som ni redan använt som skriver ut text till output.

Det är alltid bra att titta på php.net innan man litar för mycket på andra källor. Genom att kontrollera vilken version av PHP som används på din server och genom referensbiblioteket på php.net så tar du enkelt reda på vad som krävs för just din kod. I det första kursmaterielet om php så fanns en tillhörande fil som visar hur du kan generera information med funktionen phpinfo(). Denna inbyggda funktion ger en mängd information om hur PHP är konfigurerat på det aktuella systemet.

Enkel stränghantering med inbyggda funktioner
strlen($instr) - ger längden hos strängen i variabeln $instr.

trim($instr) - tar bort eventuella blanktecken i början och slutet av strängen $instr.

ltrim($instr) och rtrim($instr) Det går även att ta bort blanktecken i endast början eller slutet med ltrim respektive rtrim.

substring(instr, start, n) och
str_replace(search, repl, text) - ersätter alla förekomster av search i textstycket text med repl.

Exempel:

```
$bodytag = str_replace("%body%", "black", "<bodytext=%BODY%>");
```

ersätter alla <body text=vadsomhelst> med <body text=black>

str_ireplace() gör samma sak som str_replace, men är inte case-sensitive.

Konstanter

Ibland kan man ha nytta av att registrera ett värde som en så kallad konstant. En konstant kan man beskriva som en variabel som endast får sitt värde en gång och detta värde kan sedan inte ändras. Konstanter används ofta till att definiera max eller min-värden av någonting. Då kan man enkelt använda konstanten för att jämföra mot dessa värden och om man skulle behöva ändra max eller min-värdet så behöver man bara ändra vad konstanten sätts till på ett ställe.

Man slipper alltså gå igenom en mängd kod och leta efter så kallade hårdkodade värden och med en konstant så är det ingen risk att man råkar ändra värdet i koden.

Konstanter skapas genom den inbyggda funktionen define. Vi denna så anger du namnet på konstanten och dess värde.

```
define("MAX_PLAYERS", 6);
…
if($nbr_players<= MAX_PLAYERS){…
```

Så när vi vill använda det värde som konstanten lagra så använder vi konstantens namn istället. Konstanter blir globala så de är alltid tillgängliga.

Konstanter måste inte vara talvärden utan kan vara strängar och booleska värden också.

Namnet på en konstant följer samma regler som för variabler men skall inte inledas med ett $-tecken. Av tradition så brukar variabler namnes med stora bokstäver även om detta inte är nödvändigt. För att underlätta programmeringen så kan man ange huruvida en konstant skall vara case sensitive eller inte när den skapas genom att använda en tredje parameter för detta. Om denna sätts till TRUE så blir konsanten inte case-sensitive. Default är dock att konstanter är case sensitive.

För att kontrollera om en konstant är satt, det vill säga det redan finns en konstant med ett visst namn, så kan man använda funktionen defined. Detta underlättar om man skapar konstanter i samband med flerstegssidor.

Länkar

Generellt om cookies
http://www.php.net/manual/en/features.cookies.php

Bufferfunktioner
http://www.php.net/manual/en/ref.outcontrol.php

Det finns massvis med inbyggda funktioner på php.net under:
http://www.php.net/manual/en/funcref.php
Du förväntas inte läsa allt under denna sida men den ger dig en ide om möjligheterna i språket.

Generellt om sessions
http://www.php.net/manual/en/book.session.php

Om konstanter: Här finns även fler nyttiga funktioner att använda med konstanter
http://se2.php.net/manual/en/language.constants.php
http://se2.php.net/manual/en/function.define.php