PHP

Innehållsförteckning

Inledning 2
Ett första exempel 2
Variabler och vektorer 3
Synlighet 6
Funktioner 7
Egna funktioner 7
Felhantering 8
Databaser 9
SQL 13
Formulär och PHP 14
Cookies, inloggning och lösenordshantering 17
Sessioner 18

Inledning

PHP är ett skriptspråk som gör att vi kan föra in mer dynamik på en webbsida. Det går att skriva stora och komplexa program. Ofta hämtas informationen från en databas.

Att PHP är ett skriptspråk betyder att koden ligger inbäddad i HTML i ett dokument.
När användaren frågar efter det aktuella dokumentet (webbsidan) körs ett program på servern igång och översätter PHP-koden till det som ska göras. Det är HTML som skrivs ut till webbläsaren.

Användaren kan aldrig se PHP-koden. Användaren kan t ex inte högerklicka på sidan och be webbläsaren att visa den kod som ligger bakom. I PHP ligger koden gömd på webbservern och endast det som koden resulterar i skickas tillbaka till webbläsaren. Eftersom koden nu ligger på servern går det också att göra mycket mer saker då programmeraren via koden kan få tillgång till en kraftfull server (dator), databaser och bibliotek med funktioner i PHP.

Ett första exempel

Först tar vi en titt på hur PHP och HTML fungerar tillsammans för att sen gå in på lite mer detaljer om PHP.

Genom att bädda in koden direkt i HTML går det att på ett enkelt sätt skriva ut saker från PHP som visas på HTML-sidan. All PHP-kod måste skrivas inom dess start- och sluttagg som ser ut enligt följande:

```
<?php
  här kommer koden att ligga
?>
```

Det kan i ett dokument finnas flera segment eller kodavsnitt med PHP. Oftast ligger det då HTML mellan dessa. Vi kollar på ett första exempel som är en enkel PHP-sida med HTML och en rad PHP-kod, som skriver ut dagens datum.

```
<html>
  <head>
    <title>Dagens datum</title>
  </head>
  <body>
    <h1> Dagens datum och tid </h1>
    <?php
      echo date("Y-m-d H:i:s");
    ?>
  </body>
</html>
```

Dokumentet består alltså av en rubrik i HTML och en instruktion i PHP. PHP-koden echo date("Y-m-d H:i:s"); skriver helt enkelt ut en text till HTML-sidan innan den skickas till webbläsaren och resultatet blir en rubrik och texten dagens datum och tid under.

Resultatet som skickas till webbläsaren är följande:

```
<html>
  <head>
    <title>Dagens datum</title>
  </head>
  <body>
    <h1> Dagens datum och tid </h1>
    2004-12-02 09:12:12
  </body>
</html>
```

PHP-blocket som fanns i den efterfrågade filen på serversidan har ersatts av resultatet av den kod som låg där, dvs HTML-kod.

Variabler och arrayer

I exemplen har vi stött på något som vi kallar variabler. En variabel i ett programmeringsspråk är en behållare för ett eller flera värden. Är det ett värde säger vi helt enkelt variabel och är det flera värden säger vi att det är en array. För att fylla på med ett värde i en variabel måste man göra en tilldelning. Man tilldelar alltså ett värde till en variabel. För att göra detta användes tilldelningsoperatorn =. Ett variabelnamn i PHP börjar alltid med tecknet $ (dollar).

```
$kontaktperson = "Erik Pettersson";
```

Här tilldelas variabeln med namn kontaktperson värdet Erik Petterson, som är en sträng. Variabeln står alltid längst till vänster, följt av tilldelningsoperatorn och ett uttryck som innehåller ett värde till höger. Uttrycket i det här fallet är bara en enkel sträng men det kan också vara ett komplicerat uttryck på flera rader.

I PHP finns det tre typer av arrayer :
- Indexerade arrayer - matriser med numeriskt index
- Associativa arrayer - matriser med namngivna nycklar
- Flerdimensionella arrayer - matriser som innehåller en eller flera arrayer

Det finns två sätt att skapa indexerade arrayer. Indexet kan tilldelas automatiskt (index börjar alltid vid 0):

```
$bilar = array("Volvo", "BMW", "Toyota");
```

eller indexet kan tilldelas manuellt:

```
$bilar[0] = "Volvo";
$bilar[1] = "BMW";
$bilar[2] = "Toyota";
```

I följande exempel skapas en indexerad array med namnet $ bilar, tilldelar tre delar till den och sedan skriver ut en text som innehåller matrisvärdena:

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

`count()` funktionen används för att returnera längden (antalet element) i en array:

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

Att loopa igenom och skriva ut alla värden i en indexerad array, kan du använda en for-loop:

```
<?php
  $bilar = array("Volvo", "BMW", "Toyota");
  $arrlength = count($bilar);

  for ($x = 0, $x < $arrlength, $x++)
  {
    echo $bilar[$x];
    echo "<br>";
  }
?>
```

Associativa arrayer använder namngivna nycklar som du tilldelar dem. Det finns två sätt att skapa en associativ array:

```
$alder = array("Peter" => "35", "Ben" => "37", "Joe" => "43");

//eller

$alder['Peter'] = "35";
$alder['Ben'] = "37";
$alder['Joe'] = "43";
```

De namngivna nycklarna kan sedan användas i ett skript:

```
<?php
  $alder = array("Peter" => "35" , "Ben" => "37" , "Joe" => "43");
  echo "Peter är " . $alder['Peter'] . " år gammal." ;
?>
```

Att loopa igenom och skriva ut alla värden i en associativ array, kan du använda en foreach loop:

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

  foreach ($alder as $x => $x_value)
  {
    echo "Nyckel = " . $x . " Value = " . $x_value;
    echo "<br>";
}
?>
```

Fördjupning
http://www.php.net/manual/en/language.types.array.php

Synlighet

Variabler kan finnas på lite olika ställen på en PHP-sida. De kan ligga lokalt inne i en funktion eller på global nivå. De variabler som skapas inne i en funktion går bara att komma åt just inne i den funktionen, dvs lokalt inom funktionen. Motsatsen är en variabel som inte ligger i en funktion som kommer att vara nåbar från hela sidan, men **INTE** inne i funktioner. Synlighet brukar kallas räckvidd (eller scope) på dataspråk.
Vi tittar på tre exempel. Vad är skillnaden mellan de tre skripten? Fundera lite på varför resultaten blir så.

```
<?php
  function dummy() {
    $tal = 5;
    echo "I funktionen: " . $tal * $tal;
  }

  dummy();

  echo "<BR>";
  echo "Utanför funktionen: " . $tal * $tal;
?>
```

I funktionen: 25 Utanför funktionen: 0

```
<?php
  $tal = 5;
  function dummy() {
    echo "I funktionen: " . $tal * $tal;
  }

  dummy();

  echo "<BR>";
  echo "Utanför funktionen: " . $tal * $tal;
?>
```

I funktionen: 0 Utanför funktionen: 25

```
<?php
  $tal = 5;
  function dummy($tal) {
    echo "I funktionen: " . $tal * $tal;
  }

  dummy($tal);

  echo "<BR>";
  echo "Utanför funktionen: " . $tal * $tal;
?>
```

I funktionen: 25 Utanför funktionen: 25

Funktioner

I PHP finns en rad funktioner att använda sig av. Det är ju trots allt det som programmering handlar om, att lösa en uppgift. Och funktionerna som finns i PHP är till för att hjälpa till att lösa uppgifter som programmeraren har. Som tur är behöver du som programmerare inte hålla reda på varje funktion som finns i PHP. Vet du vilket problem som ska lösas är det ganska lätt att leta reda på de funktioner som hjälper dig med uppgiften. På PHP's webbsida finns alla funktioner samlade och namnet på en funktion säger ofta ganska väl vilket problem eller uppgift den löser. Programmering handlar inte så mycket om att skapa nya saker utan om att sätta ihop sådant som redan finns. Med andra ord, gör inte allt arbete själv som någon annan redan har gjort och som kanske t o m finns inbyggt i det språk du jobbar med

De funktioner som finns i PHP har alltså någon form av engelska namn. Ofta består namnet av en förkortning av flera ord som beskriver vad funktionen gör.

Egna funktioner

Det går också att skapa egna funktioner i PHP. Istället för att skriva flera rader kod inne i HTML-koden går det att lägga samma sak i en funktion någon annanstans i filen eller i en annan fil för att sedan bara anropa funktionen från HTML-koden. Alltså snyggare och mer lättläst. Dessutom kanske uppgiften som funktionen löser finns med flera gånger om på webbsidan. Vi slipper nu skriva ut koden för hela uppgiften flera gånger, utan anropar bara funktionen när det behövs.

Ett par saker kan vara bra att tänka på när man skriver en funktion som ska användas senare i koden eller av ett annat skript. Kommentera vad funktionen gör för att lättare kunna använda den vid ett senare tillfälle. Välj också ett passande namn redan från början. Ett passande namn beskriver på ett kort och koncist sätt vad funktionen har för uppgift. Om det blir ett för långt namn kanske det inte är din fantasi att komma på bra namn som det är fel på, förmodligen är det funktionen som gör för mycket. Det är bra om varje funktion har en avgränsad uppgift.

Tänk ut i ord vilken uppgift som du vill att en funktion eller kodsnutt ska lösa. Omvandla sedan detta steg för steg till PHP genom att slå upp vilka av PHP's funktioner som löser de delmål som varje steg i din beskrivning ska lösa.

Felhantering

Vid programutveckling finns det olika typer av fel att hantera:
Syntaxfel – felskriven kod. Programmet kan inte köras eftersom koden är skriven på fel sätt.
Logiska fel – felaktigt resultat. Programmet går att köra, men ger inte ett korrekt resultat.
Run time errors – programkörningsfel. När programmet ”kraschar”, oftast pga en oväntad förutsättning har uppstått.

För att slippa drabbas av alla fel så är det ofta nödvändigt att tillämpa defensiv programmering, dvs att i förväg förvänta sig att fel ska uppstå och i sin kod vidta åtgärder för att hantera dessa.

Man kan också ställa in olika nivåer av felhantering genom att använda `error_reporting` funktionen.

```
<?php
  //stänger av all felhantering - rekommenderas inte
  error_reporting(0);
  echo error_reporting() . "<br />";

  //normal felhanteringsnivå - inte för noga och inte för slapp
  error_reporting  (E_ERROR | E_WARNING | E_PARSE);
  echo error_reporting() . "<br />";

  //lämplig för utvecklingsmiljö
  error_reporting (E_ERROR | E_WARNING | E_PARSE | E_NOTICE);
  echo error_reporting() . "<br />";

  //alla fel och alla varningar
  error_reporting (E_ALL);
  echo error_reporting() . "<br />";
?>
```

För att slippa tala om för PHP att fel ska visas och vilken grad av felrapportering som ska användas i varje enskilt skript så går det att göra det direkt på utvecklingsservern. Efter att PHP har installerats kan detta värde ställas in i filen php.ini.

Ett typiskt nybörjarmisstag är att man skriver felmeddelanden som hjälper den som utvecklar webbplatsen, inte dess användare. Tänk noga igenom och skissa gärna upp interaktionen mellan användare och webbplats innan du börjar skriva koden. Ställ frågor som:
Vad ska hända om användaren missar att fylla i sitt namn? Vad ska hända om användarnamnet är upptaget? Och så vidare.

Bygg in dessa scenarier i systemet från början. Ett vanligt skäl till att systemet är användarovänliga är att de från början designades och konstruerades för att ha perfekta användare som aldrig gör fel och alltid noga läser alla instruktioner, kort sagt användare som tillhör en ytterst sällsynt kategori.
Folk slarvar och gör misstag när de använder ditt system och när det inte ger dem god hjälp att hantera detta så väcker det olust hos dem.

Fördjupning:
http://www.sitepoint.com/error-handling-in-php/

Databaser

En databas är en samling data som är elektroniskt lagrad och åtkomlig. Den är **persistent** (försvinner inte när man avslutar programmet eller stänger av datorn) och har en formell struktur.

En databashanterare (Data Base Managment System, DBMS) är den programvara som hanterar databaser.

Varför databas istället för filer. Det finns ganska många fördelar med att i stället använda en databashanterare. De viktigaste fördelarna är att det är

- *enkelt*
- *kraftfullt*. Att komplicerade saker går att göra på ett enkelt sätt.
- *flexibelt*. Att ett system är *flexibelt* betyder att det är lätt att ändra

Databasteknik möjliggör samtidig åtkomst av data.

Vad händer om flera personer samtidigt håller på och ändrar i kundregistret? Då är det lätt hänt att en person skriver över ändringar som en annan person gjort. Databashanteraren ser till att det inte blir några skadliga krockar.

Databasteknik möjliggör bättre säkerhet.

De flesta databashanterare har mekanismer för att ge olika användare olika rättigheter i databasen, och för att skydda data mot obehörig åtkomst. Till exempel kan man ge en användare rätt att ändra i vissa delar av databasen och att söka i andra delar, medan hon inte alls får se andra delar av **databasen**.

Relationsdatabaser
Relationsdatabashanterare (RDBMS) är den vanligaste sorten. Relationsdatabaser är flexibla och kan anpassas till de flesta behov. De är ganska bra på de mesta och är ofta den lösningen som används.

Relationsdatabaser är uppbyggda av **tabeller**.
Varje rad i tabellen kallas också en **post**.
Varje kolumn i tabellen kallas också ett **fält**. Varje fält definieras avseende värdetyp och hur stor mängd data det får innehålla.
Index påskyndar sökning efter ett urval poster eller sortering när flera poster ska visas.
Ett fält i varje tabell (eller i undantagsfall en kombination av fält) bör vara **primärnyckel** (primary key).
Primärnycklarna är också ett slags index. Primärnycklarna garanterar att varje post är unik så det får aldrig finnas två primärnycklar med samma värde.
Tabeller kan länkas till varandra genom att ett fält i en tabell matchar ett fält i en annan tabell. Fält som syftar på en primärnyckel i en annan tabell kallas för **främmande nyckel** (foreign key).
Definitionerna av tabeller, fält och index kallas för ett **schema**. Alla scheman lagras i en egen databas.
Klienter berättar för servern vad den vill göra med språket **SQL – Structured Query Language**.

En grundläggande idé i relationsdatabasen är att all data är atomär, dvs **den ska delas upp i sina minsta beståndsdelar och data ska aldrig dubbellagras**. Har du en tabell med användarna och en annan tabell med deras highscore i olika spel där varje användare kan finnas med i flera poster så ska den senare tabellen enbart ha en enda uppgift om användaren: den främmande nyckeln som gör att tabellerna kan länkas samman.

Exempel. Tabellen dubbellagrar namn och namnet har inte delats upp i förnamn och efternamn.

ID	namn	nick	spel	highscore
1	Nisse Nilsson	Nils33	Tetris	44321
2	Maria Persson	Mia44	Tetris	55374
3	Adam Andersson	Adam76	Tetris	34765
4	Nisse Nilsson	Nils33	Doom	2398
5	Adam Andersson	Adam76	Doom	4823
6	Nisse Nilsson	Nils33	Flipper	2397
7	Adam Andersson	Adam76	Flipper	5824
8	Maria Persson	Mia44	Doom	4932
9	Ada Svensson	Ada55	Tetris	52836
10	Per Persson	Perra03	Flipper	4836

Ovanstående tabell ska istället lösas med två tabeller (se nedan). I tabellen där namnet är uppdelat på förnamn och efternamn är nick primärnyckel i enligt nedan.

ID	nick	spel	highscore
1	Nils33	Tetris	44321
2	Mia44	Tetris	55374
3	Adam76	Tetris	34765
4	Nils33	Doom	2398
5	Adam76	Doom	4823
6	Nils33	Flipper	2397
7	Adam76	Flipper	5824
8	Mia44	Doom	4932
9	Ada55	Tetris	52836
10	Perra03	Flipper	4836

nick	fornamn	efternamn
Nils33	Nils	Nilsson
Mia44	Maria	Persson
Adam76	Adam	Andersson
Ada55	Ada	Svensson
Perra03	Per	Persson

Det finns formella regler för hur data ska organiseras i tabellerna. Att följa dessa regler kallas för **normalisering** och dessa regler kallas för **normalformer**.

Här följer en kort informell beskrivning av normalisering:

Varje tabell ska innehålla en slags sak. Har du två slags saker (t.ex. användare och meddelanden mellan användarna, varor och beställningar, blogginlägg och kommentarer etc) så ska du ha två olika tabeller.

Varje rad (post) i tabellen ska vara en sådan sak.

Varje kolumn (fält) i tabellen ska rymma ett atomärt värde. En kolumn för postnummer, en annan för postort, en kolumn för förnamn och en annan för efternamn etc.

Det ska finnas ett fält (eller kombination av fält) som är unikt på varje rad, en primärnyckel (primary key).

Primärnyckeln har ofta ett namn som slutar på ID. Det fältets värde kan vara skapat av databashanteraren och helt enkelt räkna upp (rad) 1, (rad) 2, (rad) 3, etc. Det kallas för autoinkrementering (inkrement = öka). Andra bra primärnycklar är användarnamn, användarens mejladresser, personnummer, ISBN-nummer på böcker EAN-koden på varor eller artikelnumret i det egna systemet etc.

Att ansluta till en databas och hämta värden

För att ansluta till en MySQL-databas från vår egen PHP-kod så bör vi välja ett PDO-gränssnittet. Det är det mest väldesignade gränssnittet och kan dessutom användas mot andra databashanterare (te.x. Oracle, MS SQL, PostgreSQL).

```
$dbh = new PDO('mysql:host=localhost;dbname=laxhjalpen;
charset=utf8','phpuser','phppass');
if (!$dbh) {
  echo "Kontakt ej etablerad";
  print_r($dbh->errorInfo());
}
else {
  echo "Kontakt är etablerad";
}
```

I stället för phpuser och phppass skriver du naturligtvis det användarnamn och passord som du valt.

Läsa poster från en ansluten databas.

```
//Fråga efter poster
$sql = "SELECT * FROM articles ORDER BY pubdate DESC LIMIT 0,5";
$stmt = $dbh->prepare($sql);
$stmt->execute();

//Läs de hämtade postern
while ($article = $stmt->fetch()) {
  var_dump($article);
}

//Fråga efter poster
$sql = "SELECT * FROM articles WHERE slug = :slug";
$stmt = $dbh->prepare($sql);
$stmt->bindParam(“:slug”, $slug);
$stmt->execute();
//Läs de hämtade postern
$blogpost = $stmt->fetch();
foreach ($blogpost as $post) {
  var_dump($post);
}
```

Fördjupning

http://www.php.net/manual/en/intro.pdo.php

SQL

SQL står för Structured Query Language.
SQL är ett standardiserat språk för att komma åt och manipulera databaser (skapa, läsa, uppdatera, radera).

Vad kan SQL göra?

- SQL kan köra frågor mot en databas
- SQL kan hämta data från en databas
- SQL kan infoga poster i en databas
- SQL kan uppdatera poster i en databas
- SQL kan ta bort poster från en databas
- SQL kan skapa nya databaser
- SQL kan skapa nya tabeller i en databas
- SQL kan skapa lagrade procedurer i en databas
- SQL kan skapa vyer i en databas
- SQL kan ställa in behörigheter för tabeller, procedurer och vyer

Även om SQL är en ANSI (American National Standards Institute) standard, det finns olika versioner av SQL-språket. Det kan alltså skilja lite beroende på databashanterare.

De flesta av de åtgärder du behöver för att utföra på en databas görs med SQL-satser.
Följande SQL-sats hämtar alla poster i "Customers" tabellen:

```
SELECT * FROM Customers;
```

Några av de viktigaste SQL-kommandon
SELECT - hämtar data från en databas
UPDATE - uppdaterar data i en databas
DELETE - raderar data från en databas
INSERT INTO - infogar nya data i en databas
CREATE DATABASE - skapar en ny databas
ALTER DATABASE - modifierar en databas
CREATE TABLE - skapar en ny tabell
ALTER TABLE - ändrar en tabell
DROP TABLE - tar bort en tabell
CREATE INDEX - skapar ett index (söknyckeln)
DROP INDEX - tar bort ett index

Fördjupning
http://www.w3schools.com/sql/sql_intro.asp

Formulär och PHP

Formulär är ett bra sätt för att kunna ta emot data från användare (klient) och skicka den vidare till en server.

Formulär är ett sätt att få information från användaren så att den går att bearbeta med ett PHP-script. Det finns två sätt att läsa av informationen från formuläret. Det ena är GET det andra är POST.

Med GET läser man det som finns i adressfönstret. Du har säkert sett webbadresser som har ett frågetecken och något annat efter. Det som kommer efter är det som man kan läsa av med PHP.

Exempel: http://localhost/welcome.php?fname=Peter&age=37

Här finns det två variabler, fname och age, som har värdena Peter och 37. Det här är ett väldigt enkelt sätt att skicka variabler mellan olika sidor. Med enkelheten kommer också riskerna. Vem som helst kan ändra variablerna och på det sättet endera hacka sig in i din databas eller bara förstöra för dig. Speciellt känslig information som lösenord skall aldrig skickas med ett GET kommando eftersom texten syns för alla.
En annan fördel är att en URL skapad med GET kan sparas som bokmärke, vilket inte en URL skapad med POST kan.

Du har en sida med ett formulär som har två textfönster och en knapp:

```
<form action="welcome.php" method="get">
  Name: <input type="text" name="fname">
  Age: <input type="text" name="age">
  <input type="submit">
</form>
```

Trycker man på knappen skickas informationen i formuläret till sidan welcome.php och den kan se ut så här:

```
<html>
  <body>
    Welcome <?php echo $_GET["fname"]; ?>!<br>
    You are <?php echo $_GET["age"]; ?> years old.
  </body>
</html>
```

Om du nu fyllde i i formuläret att du heter Peter och är 18 år gammal ser informationen i adressfältet på webläsaren:

http://localhost/welcome.php?fname=Peter&age=18

Och det som slutligen syns på sidan welcome.php är:

```
Welcome Peter!
You are 18 years old.
```

POST har mycket högre säkerhet eftersom det är du som programmerare som avgör vad som ska skickas och det går inte att lägga in andra värden lika enkelt. Ska känslig information som inloggningsnamn eller lösenord skickas mellan sidor ska alltid POST användas.
Det enda som skiljer jämfört med GET är definitionsraden i formuläret. Med POST ser samma formulär som användes i exemplet med GET ut:

```
<form action="welcome.php" method="post">
  Name: <input type="text" name="fname">
  Age: <input type="text" name="age">
  <input type="submit">
</form>
```

I welcome.php anger vi att variablerna skall läsas med POST istället för GET:

```
<html>
  <body>
    Welcome <?php echo $_POST["fname"]; ?>!<br>
    You are <?php echo $_POST["age"]; ?> years old.
  </body>
</html>
```

Och slutresultatet för användaren är likadant om samma värden används:

```
Welcome Peter!
You are 37 years old.
```

Exempel på ett enkelt formulär i HTML där även säkerhetsaspekter inkluderas.

```
<form action="action.php" method="post">
  <p>Your name: <input type="text" name="name" /></p>
  <p>Your age: <input type="text" name="age" /></p>
  <p><input type="submit" /></p>
</form>
```

```
<?php
  $name = "";
  $age = "";
  if (isset($_GET['name'])) {
    //$name = $_GET['name'];
    $name = filter_input(INPUT_GET, 'name', FILTER_SANITIZE_ENCODED,
                         FILTER_FLAG_STRIP_LOW);
  }

  if ( isset($_GET['age']) ) {
    //$age = $_GET['age'];
    $age  = filter_input(INPUT_GET, 'age', FILTER_SANITIZE_NUMBER_INT)
  }

  header("Content-type: text/html; charset=utf-8");
  echo "<p>".$name."</p>";
  echo "<p>".$age."</p>";
?>
```

Med HTML5 och JavaScript kan man kontrollera att ett formulär fyllts i på rätt sätt redan på klienten. Det har många fördelar:
Eftersom kontrollen kan köras redan under tiden formuläret fylls i så kan det ge en omedelbar feedback till användaren.
Kontrollen sker innan något skickats till servern, vilket innebär att användaren slipper vänta på den fördröjning som det innebär att data går till servern och tillbaka (roundtrip).
Sådana kontroller avlastar servern så den slipper hantera dåliga data (vinner prestanda).

Däremot innebär det inte att man slipper kontrollera data på servern. Klientkontrollen kan alltid kringgås. Det innebär att servern alltid måste göra en komplett dubbelkontroll av all indata. Man kan aldrig lita på data från användaren. Det är en regel utan undantag.
Kontroll handlar om säkerhet och användbarhet. Därför bör den utformas så att felmeddelanden hjälper användaren, inte bara utvecklaren.

Läs all indata med filterfunktionerna
Fördelarna med filterfunktionerna är många. Dels pedagogiskt tvingar de att all data betraktas som potentiellt kontaminerad och tekniskt går det att göra många sanerande och validerande åtgärder.

Get-data	filter_input(’INPUT_GET’, ’foo’, …);
Post-data	filter_input(’INPUT_POST’, ’foo’, …)
Cookie-data	filter_input(’INPUT_COOKIE’, ’foo’, …);

Fördjupning
http://php.net/manual/en/tutorial.forms.php
http://php.net/manual/en/ref.filter.php

Cookies, inloggning och lösenordshantering

Webbservrar är normalt utan minne för sina besökare. En webbläsare kan be att få en sida och när den har skickats ihop med alla sina resurser (bilder, CSS, skript, etc.) så kopplas webbläsaren bort. Webbläsaren är overksam i förhållande till servern under tiden som användaren sedan tittar på sidan och genom att därför koppla ner den så kan webbservern betjäna fler klienter. Overksamma klienter blockerar inte en anslutning och varje anslutning kräver nätverksresurser och minne.
På datorkommunikationsspråk uttrycks detta som att http är ett förbindelselöst protokoll. Till skillnad från t.ex. ftp som är förbindelseorienterat.
Det är dock vanligt att en webbplats idag vill kunna komma ihåg varje klient så att den kan möjliggöra känslan av att vara uppkopplad för användarna eller på andra sätt förbättra användarupplevelsen. Detta löser utvecklare normalt med hjälp av **kakor**, **cookies**.

En **cookie** eller **kaka** är en liten textbaserad datafil som en webbserver kan be att få spara i webbplatsbesökarens dator. Genom att kakorna i allmänhet skickas tillbaka med varje förfrågan till webbplatsen ifråga är det möjligt för servern att hålla reda på besökarens preferenser eller identitet (i den mån den är känd).

Det fungerar ungefär så här:
Webbläsaren skickar en begäran till servern, och som svar dessutom skickar med en cookie som en del av http-huvudet. När webbläsaren ber att få nästa resurs (dokument, CSS-fil, bild etc.) så skickas denna cookie med tillbaka. På så vis kan webbservern använda innehållet i cookien.

Cookies kan brukas och missbrukas. Pga möjligheten att hålla koll på användarens surfvanor är de omstridda. I sig är tekniken dock nödvändig. Det finns inga bättre alternativ som låter oss bygga inloggningssystem och liknande.

I PHP kan man ställa in cookies med funktionen **setcookie** och eftersom cookies sänds som en del av http-huvudet så måste den användas före all vanlig output, precis som funktionen header.
För att läsa cookies som har skickats finns arrayen $_COOKIE och än hellre funktionen **filter_input**. Även cookies kan användas för att attackera en webbplats så de måste hanteras som helt osäker data, precis som allt annat som kommer från användaren.

```
<?php
  setcookie("firstname", "Alex", time()+3600);
  setcookie("lastname", "Porter", time()+3600);
  echo "Skrivit en cookie";
>
```

```
<?php
  echo $_COOKIE['firstname'];
  echo $_COOKIE['lastname']; // filter_input(’INPUT_COOKIE’,  ’lastname’, …);
>
```

Fördjupning
http://sv.wikipedia.org/wiki/Cookie

Sessioner

Kommandot `session_start` måste komma före all normal output, eftersom det sätter en cookie med vår sessionsid.
Testa detta! Sessionsvariabeln (counter) lagras på servern och webbläsaren ser aldrig den, annat än som ett resultat från vår echo-sats. Men webbservern känner nu igen webbläsaren från sidvisning till sidvisning. Vi har en session.

```
<?php
  session_start();
  if ( empty($_SESSION['counter']) ) {
    $_SESSION['counter'] = 0;
  }
  $_SESSION['counter']++;

  header("Content-type: text/html; charset=utf-8");
  echo <<<HTML
  <h1>Sidan har visats {$_SESSION['counter']} gånger.</h1>
  <p>Ladda om sidan med CTRL/CMD+R eller F5.</p>
HTML;
```

På Internet skickas data med protokollet http och det var ett förbindelselöst protokoll. Detta betyder att det inte håller reda på klienten efter det att data har skickats. När nästa förfrågan kommer till servern har den ingen aning om att klienten nyss skickade en annan förfrågan. Det här är ett litet problem för programmeraren som vill hålla koll på sina användare.

Nu finns det något som kallas för sessioner i PHP som hjälper till med detta.

En session kan liknas vid en besöksbricka som du får som besökare på ett företag. Du har hela tiden med dig den som ditt identitetskort som besökare. Den fungerar som ett passerkort i en del dörrar och det kan även finnas annan information på den. När du lämnar företaget lämnar du även kvar din besöksbricka. Skulle du gå in utan att ha din besöksbricka på dig/med dig blir du inte igenkänd och insläppt på olika stället på företaget.

Samma gäller för en webbsida och där en användare loggar in och blir då registrerad i en session, som tas bort när användaren loggar ut. På det här sättet kan systemet hålla reda på om en viss användare är inloggad och då ge tillgång till sådant material som bara är för inloggade. Systemet i det här fallet är den PHP-kod som är till för att hantera besökarna på sidan. Varje besökare på en webbsida tilldelas en egen session och programmeraren kan lagra de data som är nödvändiga i sessionen. Ofta räcker det med att lagra användarens användarnamn, resten finns förmodligen att hämta hem från en databas.

Varje ny sida som ska hålla reda på en inloggning (session) måste starta sessionshanteringen först i skriptkoden. Den funktion som sköter detta är `session_start()` och måste alltså finnas med på varje sida. När sessionen är startad går det att registrera den data som ska finnas tillgänglig senare och på andra sidor.

För att lägga till variabler till sessionen används variabeln $_SESSION som är en vektor med alla sessionsvariabler i. T ex kan vi genom följande kod lägga till den användare som precis loggat in på en sida.

```
<?php
  session_start();
  $_SESSION['username'] = $_POST['username'];
?>
```

Efter detta kollar vi helt enkelt om det finns en session registrerad

```
<?php
session_start();

...

if ($_SESSION['username'])
{
  // Här kan vi t ex skriva ut ett välkomstmeddelande
  // eller hämta användarens personliga data
}
```

För att nollställa sessionen och ta bort alla dess värden behöver man göra två saker. Dels måste alla värden tömmas och det görs genom att tilldela sessionsarrayen en tom array.

```
$_SESSION = array();
```

Efter det här tar vi bort resterna av sessionen genom att använda funktionen session_destroy(). Nu är allt borttaget och sessionen går att återskapa om man så vill. För att trycka på säkerheten på sin webbsida är det också bra att verkligen kolla så att sessionen är borttagen helt och hållet.

Fördjupning

http://en.wikipedia.org/wiki/Session_(computer_science)