

Leksjon 3: Bruk av skjema

Svend Andreas Horgen, IDI Kalvskinnet, NTNU

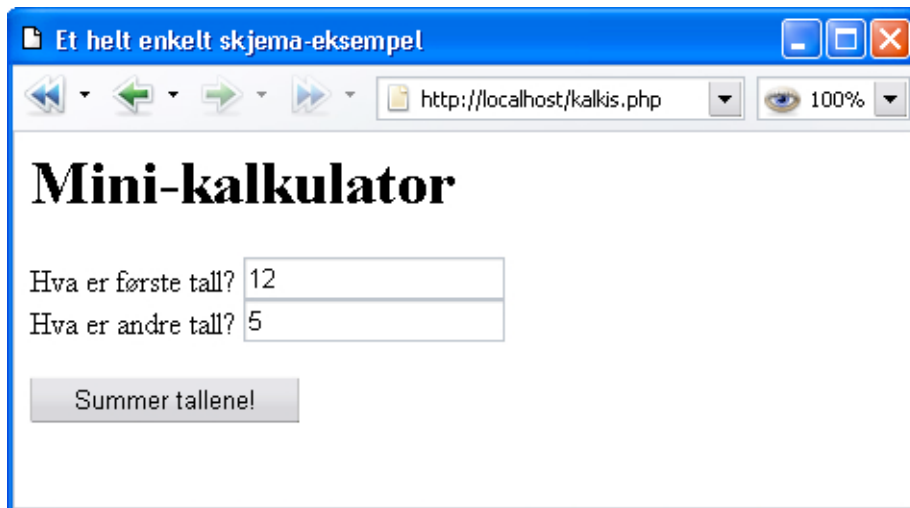
Lærestoffet er utviklet for emnet IINI3003 Webprogrammering med PHP

Resymé: Skjemabehandling er viktig i websammenheng. Det er også morsomt å kunne oppnå kommunikasjon med brukeren. Leksjonen vil se på praktisk bruk av skjema, og også gå litt dypere inn i samspillet mellom klient og tjener enn det boka gjør. I tillegg tas noe om feilhåndtering med.

1. SKJEMA	2
1.1. KODEN FOR Å LAGE SKJEMAET TIL MINI-KALKULATOREN	2
1.2. KODEN FOR Å PROSESSERE INFORMASJONEN – REGNE UT SUMMEN	3
2. HTTP (HYPERTEXT TRANSFER PROTOCOL).....	4
2.1. UTDYPENDE OM FORESPØRSELEN FRA KLIENT TIL TJENER.....	4
2.2. LITT UTDYPENDE OM SVARET FRA TJENER TIL KLIENT.....	4
2.3. SERVER-VARIABLER.....	5
3. DEBUGGING OG FEILSJEKKING.....	6
3.1. SYNTAKTISKE FEIL.....	6
3.2. LOGISKE FEIL	7
3.2.1. Dele med null.....	7
3.2.2. Uendelige løkker.....	7
3.2.3. Feil variabelnavn i for-løkker	7
3.2.4. Feil bruk av operatorer	7
3.3. STRATEGISK BRUK AV VAR_DUMP OG ECHO KAN AVDEKKE FEIL	8
3.4. HJELP, FEILMELDINGENE ER SÅ KRYPTISKE!	8
3.5. ENKEL VALIDERING AV ET SKJEMA	9

1. Skjema

Bokas kapittel 4.2 introduserer bruk av skjema, samt forskjellen på POST og GET. Før du begynner lesingen, kan du se på hvordan en enkel mini-kalkulator kan lages. Brukeren skal kunne oppgi to tall, trykke på en knapp, og så få se summen av disse. Skjermbildet kan se slik ut:



Figur 1: En enkel kalkulator

1.1. Koden for å lage skjemaet til mini-kalkulatoren

Det trengs ikke noe PHP-kode for å lage skjemaet, men det kan være greit å la filen være av type .php for å lette utvidelse i ettertid. Kildekoden ser slik ut:

```
<html><head><title>Et helt enkelt skjema-eksempel</title></head>
<body>
  <h1>Mini-kalkulator</h1>
  <form action="beregning.php" method="get">
    Hva er første tall?
    <input type="text" name="t1">
    <br>
    Hva er andre tall?
    <input type="text" name="t2">
    <p>
      <input type="submit" value="Summer tallene!">
    </form>
  </body></html>
```

Elementet `<form>` angir hvilken side som informasjonen skal sendes til, og hvordan informasjonen skal sendes. Du kan lese om forskjellen mellom GET og POST i boka etter å ha fullført dette eksempelet. Tenk da også over ulike bruksområder.

`<input>` er et mye brukt element, fordi det er neste attributt som bestemmer hvordan utseendet blir. Når `type="text"` blir resultatet en tekstboks hvor brukeren kan skrive inn informasjon. I skjermbildet er tallene 12 og 5 skrevet inn.

Tekstboksen kan om ønskelig formatteres ytterligere. Hvis for eksempel `size="3"` brukes vil tekstboksen bli mye mindre. Brukeren kan likevel skrive inn så mye informasjon som ønskelig. For å begrense til maksimalt 4 sifre, kan `maxlength="4"` brukes. Et lite aporopos: Det er også mulig å gjøre tekststørrelsen mindre og legge på skygge og farge. Da må CSS brukes eller liknende brukes.

Videre må elementenes navn være unike. Vi har kalt de to feltene for `t1` og `t2`, mens knappen ikke har fått navn. Det skyldes at knappen ikke skal brukes – den kan derfor være navnløs uten at noen merker det.

Elementet `<input type="submit">` gir en knapp som kan trykkes på. Dets value-attributt angir teksten som skal stå inne i knappen, og siden dette er bare tekst, kan du bruke mellomrom og spesielle tegn som du måtte ønsker. Knappens størrelse tilpasses til teksten.

1.2. Koden for å prosessere informasjonen – regne ut summen

Det er enkelt å summere to tall, også i PHP. Informasjonen oversendes til scriptet `beregning.php` i det knappen trykkes, og vil bli tilgjengelig i den superglobale matrisen `$_GET[]`. Navnet på hvert skjemaelement blir nøkler i matrisen og brukerens oversendte data blir tilhørende verdi.

Det vil si at `$_GET['t1']` vil ha innholdet `"12"` siden det var teksten 12 som brukeren skrev inn i det første tekstfeltet. Tilsvarende vil `$_GET['t2']` ha tekststrengen `"5"`.

Data som mottas er altså av type string. Likevel går det bra å summere to tall (en matematisk operasjon) fordi når setningen

```
$sum = $_GET['t1'] + $_GET['t2'];
```

utføres, foretas en konvertering fra string til integer, og dermed blir summeringen korrekt. Se tilbake til kapittel 2 dersom du har glemt bort dette. Hele koden for å vise resultatet, blir slik:

```
<?php
echo "  <h1>Resultat av mini-kalkulator</h1>";
$sum = $_GET['t1'] + $_GET['t2'];
echo "Summen av " . $_GET['t1'] . " og " . $_GET['t2'] . " er: $sum";
?>
```

Legg merke til hvordan spørrestrengen (det bak spørsmålstegnet) har den informasjonen som brukeren skrev inn.



Figur 2: Behandling av skjemaet: Tallene summeres

Les nå resten av kapittel 4 i boka om praktisk og mer avansert bruk av skjema.

2. HTTP (Hypertext Transfer Protocol)

I kapittel 1 i boka står det forklart hva som skjer når en bruker skriver inn en adresse i nettleseren. Kommunikasjonen er klar: Klienten ber tjeneren om å få innholdet, og tjeneren sender ønsket side tilbake, så sant dette er mulig. Det vil ofte skje en del prosessering på tjeneren i mellomtiden, men dette er usynlig for klienten.

Teorien kan virke kjedelig, men er veldig nyttig for å forstå programmering bedre. Som den italienske forskeren og kunstneren Leonardo da Vinci (1452–1519) sa det:

”Den som bare elsker praksis uten teori, er som en sjømann som seiler en skute uten ror og kompass. Han vil drive med vær og vind, uten mål og mening. Men den som har lært å seile og navigere, må også ut på havet og bruke kunnskapene sine i praksis”.

I programmering gjelder det samme. Praksis alene er ikke godt nok. Heller ikke bare teori gir den fulle forståelse. Kombinasjonen av teori og praksis er veldig viktig for å lykkes, spesielt i datafag!

2.1. Utdypende om forespørselen fra klient til tjener

Alle lenker, enten klikket på eller skrevet inn i adressefeltet, forårsaker at det sendes en GET-forespørsel til tjeneren. Sammen med forespørselen sendes også informasjon om nettleseren, og alt dette sendes i et såkalt HTTP-hode, eller en header. Gitt følgende lenke:

<http://www.iie.ntnu.no/fag/php/index.php>

Første delen av denne URL'en spesifiserer at HTTP-protokollen brukes. Neste del spesifiserer navnet på maskinen (www.iie.ntnu.no) som har denne filen. Denne maskinen har en tjener som venter på forespørsler (på TCP/IP-port 80 fordi det ikke er oppgitt noe annet). Det som mottas er for eksempel følgende:

```
GET /fag/php/index.php HTTP/1.1
Host: www.iie.ntnu.no
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1) Opera 7.54 [en]
Accept: text/html, image/gif, image/jpeg, audio/x
Accept-Encoding:gzip
```

En kort forklaring er på sin plass. Vi skal ikke gå i detalj angående dette, men det kan være greit å utvide forståelsen av hva en forespørsel innebærer på dette stadium.

- Tjeneren mottar ikke hele URL-en i første linje, for dette er ikke nødvendig.
- Tjeneren må sørge for ikke å bruke en mer avansert protokoll enn det klienten kan bruke. Hvis tjeneren sender et svar tilbake med HTTP/1.2 vil klienten få problemer fordi dette ikke støttes.
- Nettleseren identifiserer seg selv i linje 3

De formatene som nettleseren støtter visning av, oppgis til slutt.

2.2. Litt utdypende om svaret fra tjener til klient

På grunnlag av informasjonen som tjeneren mottar i denne HTTP-forespørselen sendes et svar tilbake til klienten. I de fleste tilfellene vil dette svaret inneholde et nytt HTTP-hode, samt noe

mer – nemlig innholdet i form av HTML-kode. Dersom det var andre dataformater som ble forespurt, for eksempel en fil eller en videosnutt, er det det som følger etter http-hodet.

Slik kan svaret på forespørselen se ut:

```
HTTP/1.1 200 OK
Date: Tue, 14 Aug 2004 12:05:43 GMT
Content-Type: text/html
Content-Length: 12881

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
... bla bla bla, her er resten av innholdet i websiden
</BODY>
</HTML>
```

Etter HTTP/1.1 kommer en statusmelding. Koder i området 100-199 angir at klienten må gjøre ett eller annet, altså informasjon til klienten. Koder i området 200-299 indikerer at forespørselen har gått bra. 400-499 betyr at klienten har gjort noe feil, mens 500-599 betyr at tjeneren har gjort en feil. Vi kommer tilbake til 401-feil i forbindelse med autentisering (kapittel 11 i boka).

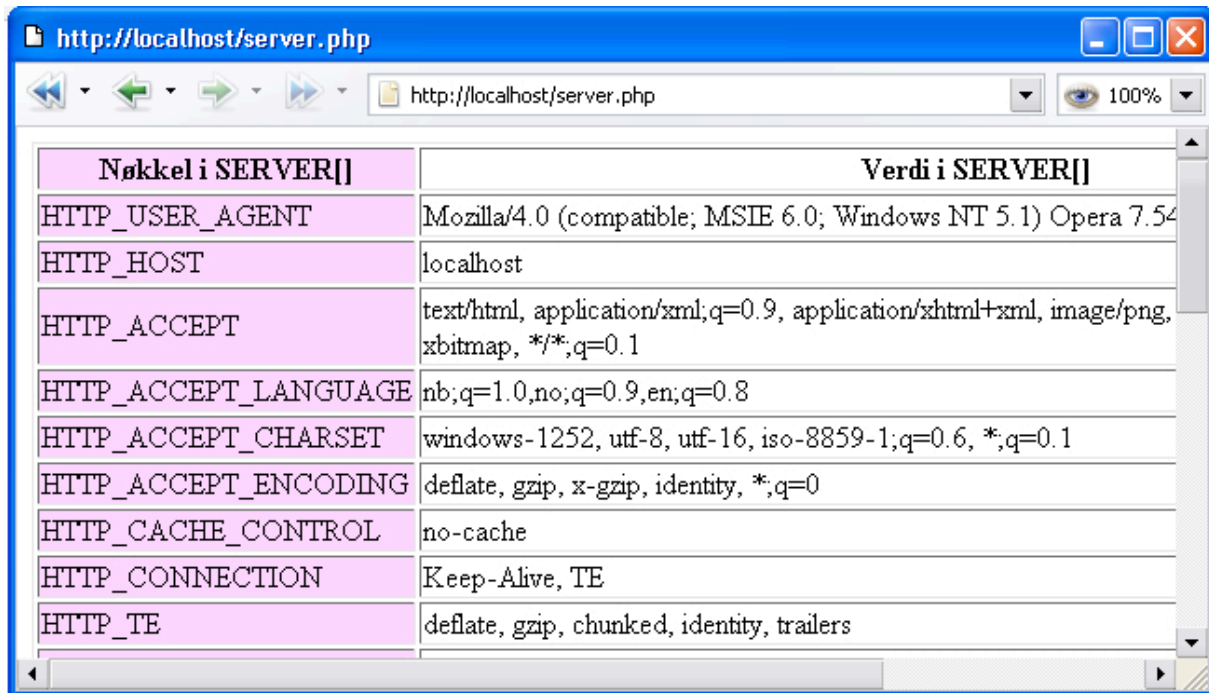
2.3. Server-variabler

Informasjonen som mottas i HTTP-forespørselen, blir tilgjengelig i den superglobale matrisen `$_SERVER[]`. Ved hjelp av en foreach-løkke og en tabell kan du skrive ut all informasjon som befinner seg i denne matrisen på en oversiktlig måte.

```
<?php /* Dette scriptet heter server.php */
//lag tabell med oversikt over nøkkel og verdi
echo "<table border='1'>";
echo "<tr><th bgcolor='#FFCCFF'>Nøkkel i SERVER[]</th>";
echo "<th>Verdi i SERVER[]</th></tr>";

foreach ($_SERVER as $nøkkel => $verdi){
    echo "<tr><td bgcolor='#FFCCFF'>$nøkkel</td>";
    echo "<td>$verdi</td></tr>";
}
echo "</table>";
?>
```

Litt lenger ned enn finner du også `REQUEST_URI`, `REQUEST_METHOD`, `PHP_SELF` og `QUERY_STRING`. Dersom du kaller siden på nytt, men oppgir `server.php?navn=ola` i stedet for bare `server.php`, vil du se at både `QUERY_STRING` og `REQUEST_URI` endrer innhold. `PHP_SELF`, derimot, er uendret, og har stien fra og med webrota. Dermed kan `$_SERVER['PHP_SELF']` med fordel brukes for å bygge opp lenker til samme side, hvis det skulle være ønskelig.



Nøkkel i SERVER[]	Verdi i SERVER[]
HTTP_USER_AGENT	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1) Opera 7.54
HTTP_HOST	localhost
HTTP_ACCEPT	text/html, application/xml;q=0.9, application/xhtml+xml, image/png, xbitmap, */*;q=0.1
HTTP_ACCEPT_LANGUAGE	nb;q=1.0,no;q=0.9,en;q=0.8
HTTP_ACCEPT_CHARSET	windows-1252, utf-8, utf-16, iso-8859-1;q=0.6, *,q=0.1
HTTP_ACCEPT_ENCODING	deflate, gzip, x-gzip, identity, *,q=0
HTTP_CACHE_CONTROL	no-cache
HTTP_CONNECTION	Keep-Alive, TE
HTTP_TE	deflate, gzip, chunked, identity, trailers

Figur 3: Utskrift av all informasjon som ligger i \$_SERVER[].

3. Debugging og feilsjekking

Det er mange kilder til feil i et program, og å finne dem er ikke alltid like enkelt. Når vi programmerer PHP i en teksteditor har vi ikke tilgang til kraftige debuggingsverktøy som ved bruk av for eksempel Visual Studio .NET for programmering i Visual Basic og C# (leses C sharp), Codewarrior for C og Java, eller Jbuilder for Java. Det fins editorer som tilbyr ulik grad av hjelp i programmeringen også for PHP, fra gratis programmer som gjør enkel bruk av fargekoder til kommersielle debuggere. Uansett hva du bruker vil du oppleve å støte på egenproduserte feil. Vi kan gruppere disse i syntaktiske og logiske.

3.1. Syntaktiske feil

For at parseren skal kunne tolke scriptene våre, kreves en høy grad av nøyaktighet av den som programmerer. De vanligste syntaktiske feilene er å:

- Glemme dollartegn foran en variabel, spesielt inne i for-løkker.
- Glemme semikolon etter en setning.
- Være unøyaktig med tanke på små og store bokstaver.
- Glemme `<?php` for å starte en kodesnutt og `?>` for å avslutte den.
- Glemme å avslutte en kontrollstruktur, en streng eller en parentes.
- Stave innebygde funksjoner litt feil eller bruke funksjoner som ikke er installerte.

Unøyaktighet i syntaksen er en ting man bare må øve seg til å se når en feilmelding kommer. Det kan være lurt å sjekke linjene før og etter, i tillegg til den linjen som feilmeldingen antyder skapte trøbbel.

3.2. Logiske feil

Dersom du først legger et tall i variabelen `$Test` og så legger et nytt tall i samme variabel ved å skrive `$test = 23;` vil du oppleve en logisk feil. Det kommer nemlig ingen feilmelding på skjermen, men den observante leseren ser at vi i stedet for én variabel, har to variabler siden `$test` og `$Test` er to ulike variabelnavn. Dette er en av konsekvensene ved at PHP ikke krever deklarerer av variablene på forhånd.

Eksempelen illustrerer en logisk feil som ikke medførte noe programkrasj. Såkalte *runtime errors* skyldes ofte logiske feil som aksepteres og utføres av PHP-tolkeren, og ender i at programmet ikke fullføres. En *fatal error* er en feil som stopper selve utførelsen av et program. Ikke alle runtime errors er fatal errors.

3.2.1. Dele med null

Det er ikke lov verken i dataverdenen eller i den matematiske, å dele et tall med null. Grunnen er at svaret ikke eksisterer. Lager vi en for-løkke som deler tallene 7, 6, 5, 4, osv på 3 er det fort gjort å ikke stoppe før 0 men på 0, hvilket medfører feilen "division by zero".

3.2.2. Uendelige løkker

Dette er gjennomgått og eksemplifisert i kapittel 3.3.4 i boka, men gjentas her kort:

```
$teller = 0;
while ($teller < 10) {
    echo "Tallet er $teller";
}
```

Løsningen er å enten sette `++$teller < 10` inne i betingelsen, eller å sette `$teller++;` som en ny linje etter `echo`-setningen inne i løkken. Det siste er mest vanlig og oversiktlig, men det er ingenting i veien med å gjøre det kort og enkelt.

3.2.3. Feil variabelnavn i for-løkker

Ser du hva feilen i følgende formulering er?

```
for($i=0; i<13;i++);
```

Det er fort gjort å glemme at *dollartegnet* må med for alle variabler. Denne type feil kan være vanskelig å oppdage.

3.2.4. Feil bruk av operatorer

Det er fort gjort å bruke tilordningsoperatoren i stedet for sammenlikningsoperatoren uten å oppdage det, siden de har så lik syntaks. I neste kodesnutt vil if-delen **alltid** utføres. Uttrykket `$lykkeTall = 12` vil nemlig bestandig være mulig å utføre for tolkeren, da det ikke er noe galt i å tilordne verdien 12 til en variabel, heller ikke som del av et uttrykk inne i en betingelse. Det vi mente å skrive var *to* likhetstegn for å teste hvorvidt verdien i `$lykkeTall` er 12 eller ikke. Feil som innebærer at vi glemmer ett enkelt tegn er vanligere enn en skulle tro, og dessuten vanskelig å oppdage.

```
// $lykkeTall hentes fra et skjema, anta at det er 16
if ($lykkeTall = 12) {
    echo "Du har samme lykketall som forfatteren, nemlig $lykkeTall";
}
```

```
else {
    echo "Ditt lykketall er $lykkeTall, et nokså bra valg";
}
```

Det anbefales for øvrig å alltid bruke klammeparenteser { og } til å angi starten og slutten av strukturer, selv om det ikke er nødvendig i tilfeller der det bare er én setning som skal utføres. Koden over kunne vært skrevet slik:

```
if ($lykkeTall = 12)
    echo "Du har samme lykketall som forfatteren, nemlig $lykkeTall";
else
    echo "Ditt lykketall er $lykkeTall, et nokså bra valg";
```

Det en risikerer nå, er å legge til nye setninger i ettertid, uten å huske klammene. Boka vil av og til unngå bruk av klammer, mest motivert i å spare plass.

3.3. Strategisk bruk av var_dump og echo kan avdekke feil

En mulighet for å oppdage feilen med lykketall er å skrive ut innholdet av variabelen \$lykkeTall på flere strategiske steder i koden. Bruk enten `echo $lykkeTall;` eller `var_dump($lykkeTall);` så merker du fort hvor feilen ligger. Innskuddene er vist i grått, og kan kommenteres eller slettes når feilen er oppdaget og rettet.

```
echo $lykkeTall . "<br>";
if ($lykkeTall = 12) {
    echo "variabelens verdi inne i if, er $lykkeTall <br>";
    echo "Du har samme lykketall som forfatteren, nemlig $lykkeTall";
}
else {
    echo "variabelens verdi inne i else, er $lykkeTall <br>";
    echo "Ditt lykketall er $lykkeTall, et nokså bra valg";
}
```

Før if-setningen ser vi at verdien av lykketallet er 16, mens det er blitt 12 inne i løkken når neste echo-setning utføres. Uansett hvor mye vi strever vil vi aldri få fram else-meldingen. Det bør ringe en bjelle nå om at feilen ligger mellom første og andre echo-setning.

Bruk av `var_dump` i stedet for `echo` anbefales i forbindelse med matriser. Prøv gjerne selv hva som skjer om du kjører den neste kodebiten med og uten bruk av pre-taggene:

```
echo "<pre>";
$matrise = array("Kari", 1213, "navn" => "Ola", "pinkode" => 8273 );
var_dump($matrise);
echo "</pre>";
```

3.4. Hjelp, feilmeldingene er så kryptiske!

Feilmeldinger kan synes kryptiske, men du kan også få en del informasjon fra dem og hjelp til å lokalisere og løse problemet.

Glemmer du å avslutte en parentes, får du kanskje følgende feilmelding:

```
Parse error: parse error, unexpected $end in test.php on line 26
```


Vi har brukt skjema til å sende data til et script for videre prosessering. I scriptet under er det en skrivefeil som forårsaker feil:

```
1.  if (!isset($_GET['knapp'])) {
2.      echo "<form action=\"\" . $_SERVER['PHP_SELF'] . \"\" method=get>";
3.      echo "<input type=\"text\" name=\"navn\">";
4.      echo "<input type=\"submit\" name=\"knapp\">";
5.  }
6.  else {
7.      echo $_GET['Navn'];
8.  }
```

Feilmeldingen er:

```
Notice: Undefined index: Navn in lek02\kode\test.php on line 7
```

Leser du feilmeldingen nøye ser du at det er noe galt med indeksen til matrisen `$_GET` på linje 7. Det står at Navn er en udefinert indeks, men det kan jo ikke stemme for du husker fra koden at du kalte feltet akkurat dette. Går du derimot tilbake til URL-en vil du se at feltet egentlig heter navn med *liten* n, og det er her feilen ligger. PHP er nøye på bruk av store og små bokstaver i variabler og nøkler.

Det skal *ikke* refereres til superglobale/forhåndsdefinerte variabler inne i en tekststreng. Dette gjelder også konstanter og matriser. Dessverre er dette fort å glemme ved å skrive for eksempel

```
echo "Navnet er $_GET['Navn'] og alderen er $_GET['alder']";
```

Hvis en ikke skjønner feilmeldingen er det kanskje nærliggende å tro at det er noe feil med overføringen av skjemainformasjonen. Feilmeldingen ser slik ut:

```
Parse error: parse error, unexpected T_ENCAPSED_AND_WHITESPACE,
expecting T_STRING or T_VARIABLE or T_NUM_STRING
```

Det er ikke alltid riktig linjenummer som blir referert i feilmeldingen. Glemmer du et anførselstegn på slutten av en print-setning slik:

```
1.  <?php
2.      echo date('F');
3.      echo "hei";
4.  ?>
```

får du denne feilmeldingen, som tydelig ikke hjelper deg til å finne fram til riktig linjenummer:

```
Parse error: parse error, unexpected $end in test1.php on line 4
```

3.5. Enkel validering av et skjema

Vi skal se mye på validering av data som kommer fra brukeren utover i kurset. Bokas kapittel 11 setter fokus på sikkerhet, og de fleste kapitler har med små drypp med godbiter en kan ta med seg.

I bokas kapittel 4.2.1 er det vist et skjema med personlig informasjon som skal oversendes. Postnummer og sted, navn og favorittfarge. Hva skjer om brukeren skriver inn noe tull i stedet for et gyldig postnummer? Resultatet blir for eksempel at teksten

” Du, Svend Andreas, du bor i Trondheim med postnummer **nugatti**”. Hva så? Brukeren får vel ta konsekvensen av det som skrives inn? Det er mange situasjoner hvor informasjonen ikke bare skal vises til brukeren i en oppsummering, men brukes i videre databehandling.

Postnummeret skal kanskje være grunnlag for å sende et brev eller en pakke med vanlig post.

Feil postnummer kan unngås ved å teste om innskrevet verdi er mindre enn 9999 og større enn 1000 – i så fall vil det være et gyldig postnummer. Hvis ikke må brukeren få beskjed om at noe er galt fatt:

```
<body bgcolor="<?php echo $_GET['fargeValg']; ?>">
<h2>Velkommen <?php echo $_GET['innskrevetNavn']; ?></h2>
<?php
if ($_GET['postnummer'] > 1000 && $_GET['postnummer'] < 9999){
    echo "Du, ";
    echo $_GET['innskrevetNavn']; //kan bruke fnutter...
    echo ", du bor i ";
    echo $_GET["poststed"]; //...eller anførselstegn
    echo " med postnummer <strong>";
    echo $_GET['postnummer'];
    echo "</strong>";
} //alt gikk bra med postnummeret
else {
    echo "Feil postnummer, ";
    echo "<a href='JavaScript:history.go(-1);'>gå tilbake</a>";
    echo " og fyll ut manglende informasjon";
}
?>
</body>
```

Koden bør håndtere feil på et tidligst mulig stadium. Kapittel 11 i boka viser hvordan JavaScript kan brukes for å håndtere feil før informasjonen forlater klienten, og tar også opp hvorvidt dette er tilstrekkelig. Vi skal komme (grundig) tilbake til sikkerhet senere.