

Leksjon 4: Strukturering av kode, dine egne funksjoner

Svend Andreas Horgen, IDI Kalvskinnet, NTNU
Lærestoffet er utviklet for emnet IINI3003 Webprogrammering med PHP

Resymé: Denne leksjonen kan virke kort, og går blant annet grundig gjennom en av oppgavene i boka. Det er likevel mye arbeid knyttet til denne leksjonen – en del å lese i boka, og en del å gjøre i øvingen. Funksjoner og strukturering av kode er et viktig tema som krever noe forståelse. Har du programmert mye tidligere er det kurant, men hvis du er relativt fersk innen programmering bør du bruke en del tid på å forstå dette stoffet. Ikke bli demotivert om det virker for vanskelig, kom da heller tilbake til stoffet om noen uker.

1. INKLUDERING AV FILER.....	2
2. FUNKSJONER DU SELV LAGER.....	2
2.1. AREALET AV ET REKTANGEL.....	3
2.2. OVERFLATEAREAL = BRUK FUNKSJONEN I VIDERE BEREKNINGER	4
2.3. VALGFRIE ARGUMENTER	4
2.4. FINN VOLUMET AV EN REKKE KLOSSER.....	4
2.5. GJELDER DET SAMME FOR SIRKEL?	5
3. GJENNOMTENKTE FUNKSJONER KAN SPARE DEG FOR TID	5
4. NOEN AVSLUTTENDE ORD OM STRUKTURERING	7

1. Inkludering av filer

Boka tar i kapittel 5 for seg hvordan en informasjonsportal om solsystemet kan ta hensyn til hvem brukeren er før informasjon presenteres på skjerm. Slik opptrer portalen som ekstra brukervennlig, noe som jo må sies å være et av målene med webutvikling. For å få til den tilsynelatende intelligente presentasjonen av vårt solsystem, brukes inkludering av filer. Dersom du har bok bør du lese det eksempelet.

Kort oppsummert fungerer inkludering slik:

- Enhver fil med tekst eller kode kan inkluderes i et PHP-script med setningen `include "filnavn";`
- Innholdet i den filen som blir inkludert, settes inn akkurat der hvor include-setningen forekommer.
- PHP avslutter PHP-modus i det en fil inkluderes. Det betyr at eventuell PHP-kode (i filen som inkluderes) som starter med `<?php` og slutter med `?>` vil tolkes og utføres. Det samme vil HTML-kode.
- Dette gjelder også tekstfiler som har etternavn .txt. Ved inkludering settes innholdet rett og slett inn – uavhengig av filtype. Dermed gir det ikke mening å inkludere et bilde eller andre ikke-tekstlige formater.
- Det er vanlig å inkludere topp- og bunntekster, for å lette vedlikeholdet.
- En smart bruk av inkludering kan redusere tiden det tar å utvikle et nettsted.
- Kombinert med funksjoner er inkludering et sterkt verktøy: Egendefinerte funksjoner kan samles i en include-fil og dermed brukes over et helt nettsted.
- Når inkludering nøstes (en fil inkluderes i en som allerede er inkludert i en annen) må en holde tunga rett i munnen. Et faremoment er at stiene til ressurser blir feil, noe som kan gjøre at bilder ikke vises eller stilark ikke synes.
- Det er mulig å inkludere filer som ligger utenfor webrota, fordi PHP har tilgang til hele tjeneren, mens en besøkende bare har tilgang til det området som ligger under rota til nettstedet. Filer som inneholder brukernavn og passord bør dermed legges utenfor rota og inkluderes inn i ønsket script. Tenk over hvorfor og hvordan dette gir bedre sikkerhet.

Eksemplene i boka dekker det du trenger å vite om inkludering. Legg spesielt merke til hvordan problemet nøsting av inkluderte filer kan løses ved å bruke `include_once` / `require_once` først i stinavnet.

Du blir utfordret til å trene deg på inkludering av filer i øvingen. Trenger du mer lærestoff og ikke har boka) så fins det mange veiledere på web. Søk etter ”include php tutorial”.

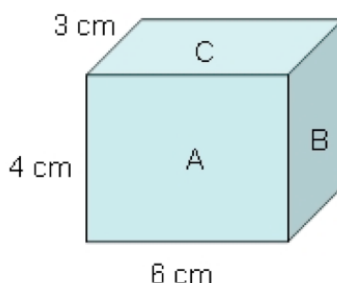
2. Funksjoner du selv lager

Egendefinerte funksjoner er hendige, og enkle å lage. I PHP kan funksjoner brukes både til å returnere verdier og skrive ting direkte til nettleseren. I websammenheng er det ofte repetisjon av oppgaver, og når en oppdager mønstre er det lurt å lage egne funksjoner. Løkker vil også

kunne passe bra ved repetisjon av oppgaver, for eksempel når en tabell med nokså like rader og kolonner skal lages. Gevinstene ved funksjoner er likevel mange. Her er noen fordeler:

- Økt mulighet for gjenbruk.
- Modularisering av kode gjør at detaljene skjules. Dette gjør at det blir lettere å lese koden ved feilleting og videreutvikling av dine script. Du kan lese et funksjonsnavn og vite hva som skjer uten at du trenger å lese alle linjene som utgjør funksjonen.
- Endring i en funksjon vil få virkning for alle steder hvor funksjonen brukes.
- Muligheten for å angi valgfrie argumenter (optional arguments) gjør funksjoner enda mer fleksible i bruk.

Nå følger en gjennomgang av oppgave 5-2 i boka, med utdypende forklaringer. Dette kan leses og forstås selv om du ikke har bok. Her skal vi se nærmere på en kube.



2.1. Arealet av et rektangel

En funksjon som finner arealet av et rektangel, må ha et navn og to argumenter med informasjon om hva lengden og bredden er. Definisjonen kan se slik ut:

```
function finnAreal($lengde, $bredde){

}
```

Oppgaven skal være å finne arealet av et rektangel, og det gjøres ved å multiplisere lengden og bredden. I stedet for å skrive ut svaret med en echo-setning, slik:

```
function finnAreal($lengde, $bredde){
    $svar = $lengde * $bredde;
    echo $svar;
}
```

velger vi å returnere svaret – dermed er det opp til det stedet hvor funksjonen kalles å bestemme hvordan utskriften skal skje – eller om svaret i det hele tatt skal skrives ut. Kanskje skal det inngå i en beregning i stedet? *Retur av verdi er mye mer fleksibelt:*

```
function finnAreal($lengde, $bredde){
    $svar = $lengde * $bredde;
    return $svar;
}
```

Den siste varianten av funksjonen kan brukes slik (antar at funksjonen er definert lenger opp i scriptet, eller i en fil som inkluderes).

```
echo "Side A er " . finnAreal (4, 6) . " cm<sup>2</sup><br>";
echo "Side B er " . finnAreal (4, 3) . " cm<sup>2</sup><br>";
echo "Side C er " . finnAreal (3, 6) . " cm<sup>2</sup><br>";
```

HTML-taggen `<sup>` betyr superscript og gir hevet/opphøyet tekst. Dermed står det cm^2 i nettleseren som følge av `cm²`. Videre kan du ikke bruke en funksjon inne i en tekststreng, men må avslutte tekststrengen først.

```
echo "Volumet av kuben i boka er ";
$volum = finnAreal(3,4) * 6;
echo "<strong>$volum</strong>";
```

2.2. Overflateareal = bruk funksjonen i videre beregninger

Kuben som er vist i boka har et totalt overflateareal som tilsvarer summen av alle flatene. Husk at en kube har to og to flater som er like. Selv om bare tre flater synes er det altså totalt 6 flater. Totalsummen blir:

```
echo "Totalsummen blir: ";
$tot = finnAreal(3,4) * 2 + finnAreal(3,6) * 2 + finnAreal(4,6) * 2;
echo $tot . " cm<sup>2</sup>";
```

Vi ser her at det er lurt at funksjonen returnerer en verdi i stedet for å skrive ut.

2.3. Valgfrie argumenter

Boka går gjennom bruk av valgfrie argumenter. Vi har nå sett at funksjonen er generell – den kan brukes i mange sammenhenger. For å finne volumet måtte vi derimot multiplisere arealet med en tredje lengde. Med (en smart) bruk av valgfrie argumenter kan funksjonen lages så generell at den både kan finne arealet av et rektangel og volumet av en kube.

```
function beregn($tall1, $tall2, $tall3=1){
    //Setter tall3 = 1 for å beregne areal
    //dersom volum-parameter er utelatt
    return($tall1 * $tall2 * $tall3);
}
```

Du tenker kanskje at det samme kunne du gjort ved å bruke if-else i funksjonen, og slik unngå valgfrie argumenter? Delvis riktig, men da tvinger du til at funksjonen alltid må kalles med tre argumenter. Når en skal finne areal er det unaturlig å kalle med et tredje dummy-argument. Løsningen over er derfor best. Funksjonen kan nå brukes slik:

```
echo "Side A er " . beregn(4, 6) . " cm<sup>2</sup><br>";
echo "Side B er " . beregn(4, 3) . " cm<sup>2</sup><br>";
echo "Side C er " . beregn(3, 6) . " cm<sup>2</sup><br>";
echo "Volumet er " . beregn(4, 6, 3) . " cm<sup>3</sup><br>";
```

2.4. Finn volumet av en rekke klosser

Neste oppgave ber om at volumet av 30 likesidede klosser skal finnes, og presenteres for eksempel slik:

```
Volumet av en likesidet kloss med sider på 1 cm er 1 cm3
```

Volumet av en likesidet kloss med sider på 2 cm er 8 cm³
 Volumet av en likesidet kloss med sider på 3 cm er 27 cm³
 Volumet av en likesidet kloss med sider på 4 cm er 64 cm³
 Volumet av en likesidet kloss med sider på 5 cm er 125 cm³
 Volumet av en likesidet kloss med sider på 6 cm er 216 cm³
 Volumet av en likesidet kloss med sider på 7 cm er 343 cm³
 ... og så videre, helt til klossen med side 30 er nådd.

Spørsmålet er hvor mange kodelinjer som må til. Er det:

- Minst 30 fordi volum-funksjonen må kalles 30 ganger?
- Mindre? Kan du gjette hvor mange?

Funksjoner kan kalles i en for-løkke, og det er trikset i dette tilfellet.

```
for ($i = 1; $i<=30; $i++) {
    echo "Volumet av en likesidet kloss med sider på $i cm er ";
    echo beregn($i, $i, $i) . " cm<sup>3</sup><br>";
}
```

Egentlig er to setninger nok, dersom echo-setningen skrives på en linje og klammene { } i for-løkken utelates. Det som er verdt å merke seg er at variabelen `$i` brukes som argument, og denne økes for hver gang av løkken.

2.5. Gjelder det samme for sirkel?

Siste delspørsmål går på om det er mulig å lage en funksjon som finner både areal og volum av en sirkel/kule.

Funksjonen under bruker en if-else struktur for å skille mellom kule og sirkel. Konstanten `M_PI` er innebygd i PHP og har den matematiske verdien til pi.

```
function rundGjenstand($radius, $kule = "nei"){
    if ($kule == "ja")
        return 4 * (M_PI * $radius * $radius * $radius) / 3; //kule
    else
        return M_PI * $radius * $radius; //sirkel
}
echo rundGjenstand(10); //areal av en sirkel med radius 10
echo "<p>";
echo rundGjenstand(10, "ja"); //areal av en sirkel med radius 10
```

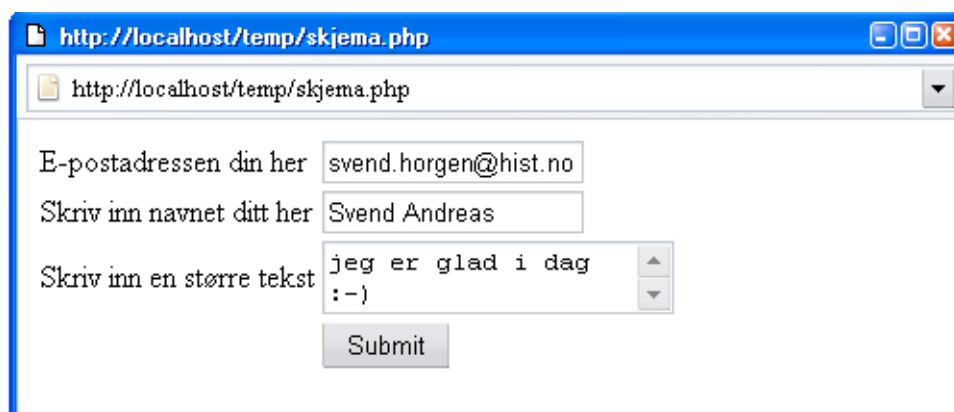
3. Gjennomtenkte funksjoner kan spare deg for tid

Du kan lage funksjoner av hva du vil. Dersom du gjør dette på en gjennomtenkt måte, kan du spare tid og øke effektiviteten på det du utvikler. Her er et eksempel på en funksjon som ikke nødvendigvis er optimal eller hundre prosent ”etter læreboka”, men som illustrerer konseptet og kanskje setter i gang tanker om hva som er mulig med litt kunnskap om egendefinerte funksjoner.

For å få presentert skjema på en ryddig måte kan det være greit å bruke en tabell med to kolonner: En kolonne med en beskrivelse av hva som skal gjøres, og en med skjemaelementene, for eksempel tekstfelt.

```
<form action='visinfo.php'><table>
  <tr>
    <td>E-postadressen din her</td>
    <td><input type='text' name='email'></td>
  </tr>
  <tr>
    <td>Skriv inn navnet ditt her</td>
    <td><input type='text' name='navn'></td>
  </tr>
  <tr>
    <td>Skriv inn en større tekst her</td>
    <td><textarea name='myeMerTekst'></textarea></td>
  </tr>
  <tr>
    <td></td>
    <td><input type='submit' name='knapp'></td>
  </tr>
</table></form>
```

Resultatet blir pent å se på:



Figur 1: Med en tabell kan du kontrollere at ting plasseres oversiktlig ved å bruke kolonner og rader

Det kan være greit å lage grensesnittet 100% i HTML, særlig hvis en grafisk designer gjør jobben med å lage utseendet. Sett at du likevel ønsker å lage grensesnittet med PHP-kode, og ofte gjør dette. Da kan følgende funksjon være grei å bruke:

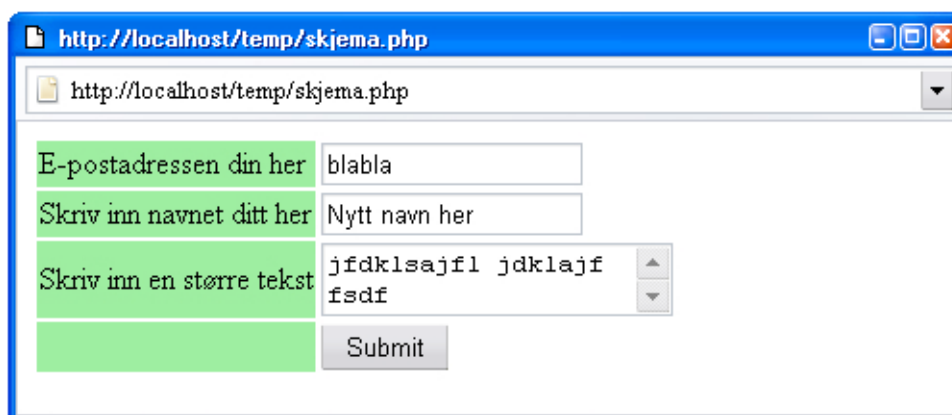
```
function lagTabell($matrise, $border=false){
  echo "<table";
  if ($border) echo " border='1'";
  echo ">\n"; //har nå lagd en tabell enten med eller uten ramme.
  foreach ($matrise as $venstre => $hoyre) {
    echo "\t<tr><td>$venstre</td><td>$hoyre</td></tr>\n";
  } //slutt gå gjennom hele matrisen og lage tabell
  echo "</table>";
} //slutt lagTabell
```

Denne funksjonen tar utgangspunkt i en matrise, og legger tekst og skjemaelementer i to kolonner. Høyre kolonne får teksten, og er i praksis det som ligger i matrisens neste nøkkelement, mens venstre kolonne får skjemaet, som er verdien til matriseelementet. Hvis du skal lage skjemaet over med denne funksjonen, må du først lage en matrise og så kalle funksjonen, slik:

```
$matrise = array (
    "E-postadressen din her" => "<input type='text' name='email'>",
    "Skriv inn navnet ditt her" => "<input type='text' name='navn'>" ,
    "Skriv inn en større tekst" => "<textarea name='mye'></textarea>" ,
    "" => "<input type='submit' name='knapp'>"
); //slutt på matrise over felt som det skal lages tabell av
echo "<form action='visinfo.php'>";
lagTabell($matrise); //lager tabell uten ramme (border=false i andre arg)
echo "</form>";
```

Denne funksjonen kommer til sin rett hvis du legger den i en include-fil og bruker den ofte. Du vil derimot se at det er vanskelig å legge til mer avanserte skjema-elementer på denne måten, og nøklene i en matrise må alltid være unike, så det er kanskje ikke den ideelle formen å bruke en slik representasjon. Likevel kan du over et helt nettsted enkelt forandre alle skjema ved å bare endre ett sted, nemlig i funksjonen. Skal du for eksempel ha grønn bakgrunnsfarge på hjelpeteksten i alle skjema, er det bare å endre til dette i funksjonen:

```
echo "\t<tr><td bgcolor='lightgreen'>$venstre</td>
      <td>$hoyre</td></tr>\n";
```



Figur 2: Endringer kan raskt gjøres om gjennomtenkte funksjoner ligger i include-filer.

Du ser selv at siste linje ikke ble helt som forventet. Dette kan enten løses ved å legge til en `if` i funksjonen, eller ved å ikke legge inn selve skjemaknappen i matrisen (slik som gjort med `<form>`).

4. Noen avsluttende ord om strukturering

Fra en programmerers ståsted er det viktig å strukturere koden i forbindelse med utvikling i PHP. For å generalisere kode som brukes ofte kan vi lage funksjoner, noe som passer når innholdet har noe variasjon fra gang til gang. Inkludering av filer passer best når innholdet er statisk.

Det er viktig å skille logikk fra presentasjon i webprogrammering. I dette emnet ser du noen ganger eksempler som ikke er helt etter boka på dette området. Årsaken er at skillet mellom logikk/design krever en ekstra grad av forståelse og modenhet, og det er ikke pedagogisk riktig å lage komplekse løsninger som i seg selv er vanskelige å forstå, når poenget er å introdusere nye konsepter som for eksempel databaseoppkobling, sikkerhet, cookies og liknende. Ha dette poenget i bakhodet når du ser kode som du kanskje mener kunne vært bedre strukturert, og ta det heller som en ekstra utfordring å prøve å omstrukturere så koden blir bedre.

Mange liker å programmere med en objektorientert tilnærming (OOP). Etter at PHP5 kom ble OOP-støtten i PHP veldig god. OOP tas opp i kapittel 13 i boka og er orienteringsstoff i dette emnet. Det fine med PHP er at du kan programmere både objektorientert og proseduralt om hverandre. Det gjør terskelen for å komme i gang liten, samtidig som brukere med mer avanserte behov kan lage kraftige OOP-baserte løsninger med stor grad av gjenbruk. Er en ekstra avansert, vil en kanskje foretrekke å bruke Model View Controller (MVC), noe som er langt utover pensum i dette emnet, men verdt å merke seg og se nærmere på for den med ambisjoner om å programmere mye PHP. For å forstå MVC må du først forstå grunnleggende OOP.

En avansert bruk av funksjoner kan være hendig, men bør benyttes med måte. Vi har tatt med noen avanserte muligheter for å gi en dypere forståelse av hvordan funksjoner fungerer og hvilke feller en kan gå i. Husk at selv om du forstår de avanserte mulighetene er det ikke sikkert at neste person som jobber på det prosjektet du benytter slike i, er like interessert i eleganse. Ofte er det enkle best, men dette er selvsagt situasjonsavhengig.

Det er også en rekke innebygde funksjoner i PHP. Når du står ovenfor et problem, kan det være naturlig å lage en funksjon for å løse problemet. Ta gjerne en titt i PHP-manualen først. Det er en viss sannsynlighet for at det allerede er laget kode for å løse problemet. Et eksempel er matriser (arrays). Det fins en rekke funksjoner for å behandle disse, for eksempel å sortere, finne differansen mellom matriser, hente ut alle nøkler og liknende.