Institutt for informasjonssystemer

Kontrollstrukturer

IS-115 Webprogrammering i PHP | Peter André Busch

Innhold

Leksjon 3: kontrollstrukturer

Du har nå fått en grunnleggende kunnskap om hvordan PHP fungerer. I denne modulen skal vi lære om kontrollstrukturer i PHP. Disse kontrollstrukturene er viktige for å kunne sjekke input og utføre handlinger deretter. Alle språk har sine variasjoner i hvordan de utfører disse kontrollene. Vi skal se nærmere på hvordan dette gjøres i PHP i denne leksjonen. Når du programmerer i PHP, kommer du fort borti behovet for disse kontrollstrukturene. Kapittel 3 i boka må leses selv om du har programmert tidligere, for denne kunnskapen er helt sentral i programmering.

1. Valgsetninger

Den ene kategorien av kontrollstrukturer, er kontroller som utfører lineære kontroller. Dersom et uttrykk er sant eller usant, vil kontrollen utføre en forutbestemt handling. Den andre kategorien kontrollstrukturer er løkker som (i utgangspunktet) itererer helt til et uttrykk tilfredsstiller exit-kriteriet (se nedenfor). For en oversikt over alle kontrollstrukturer, se php.net sine websider: https://www.php.net/manual/en/language.control-structures.php

1.1 If-kontroll

If-setninger sjekker om et uttrykk er sant, for eksempel om en variabel `§alder` er større enn 18. Vurderingen som if-løkken gjør resulterer enten i at uttrykket er «sant» eller «usant». Dersom det er «sant» utfører den noe du har bestemt. Hvis det er usant, kan enten if-setningen avsluttes direkte eller gjøre noe annet. Her er et eksempel på en if-løkke som avsluttes dersom argumentet ikke er «sant»:

```
<?php
    $dagen_i_dag = date("l");
    // Merk: Argumentet er en liten L, ikke et ett-tall
    // date()-funksjonen er en innebygd funksjon i PHP

    echo "<h1>Today is a <strong";
    if ($dagen_i_dag == "Sunday") // Legg merke til dobbelt ==
    {
        echo " style='color:blue;cursor:help'";
    }
   // Slutt på if-kode som kun utføres hvis det er søndag
   // Her legges CSS til HTML-koden i tilfelle argumentet er sant

    echo ">$dagen_i_dag</strong></h1>";
?>
```

Det som skjer her er at variabelen `§dagen_i_dag` settes til riktig ukedag ved hjelp av en innebygd funksjon i PHP (engelsk er standard, men det er mulig å endre dette til norsk – mer om det under innebygde funksjoner). Deretter skriver PHP ut deler av HTML-kode. Før resten av HTML-koden fullføres sjekkes det hvilken ukedag det er. Dersom det er søndag, vil CSS-koden ovenfor legges til (noe som gjør at teksten blir blå og at markøren endrer seg). Uansett om det er søndag eller ikke avsluttes HTML-koden med teksten i siste setning. Dette scriptet forandrer altså ukedag-teksten basert på hvilken ukedag det faktisk er. PS! Dersom du er nysgjerrig på hvordan date-funksjonen fungerer, kan du slå den opp på php.net: https://www.php.net/manual/en/function.date.php (du lærer mer om dette senere). Jeg testet scriptet på en mandag og det så slik ut:

Today is a Monday

Du kan teste mange forskjellige uttrykk med `if` (for eksempel om `§a` er større enn `§b`, om `§b` er satt til en verdi [hva som helst] eller om `§a` ikke er lik én bestemt verdi). `If`-kontrollen kan utvides til å sjekke flere uttrykk med bruk av `elseif`. Dersom vi tar utgangspunkt i scriptet ovenfor, kan dette endres til noe slikt som dette:

```
<?php
    $dagen_i_dag = date("l");
    echo "<h1>Today is a <strong";
    if ($dagen_i_dag == "Sunday")
    {
        echo " style='color:blue;cursor:help'";
    }
    // Slutt på if-kode som kun utføres hvis det er søndag
    // Her legges CSS til HTML-koden i tilfelle argumentet er sant
    elseif ($dagen_i_dag == "Monday")
    {
        echo " style='color:red;cursor:help'";
    }
    // Slutt på if-kode som kun utføres hvis det er mandag
    // Her legges CSS til HTML-koden i tilfelle argumentet er sant

    echo ">$dagen_i_dag</strong></h1>";
?>
```

1.2 Switch-kontroll

En annen kontroll er `switch()`-setningen. Den ligner på en serie if-setninger om det samme uttrykket. Forskjellen mellom en rekke if-setninger og `switch()`-setningen er at uttrykket du sammenligner med, blir evaluert bare én gang i en `switch()`-setning. I en god del tilfeller kan det være lurt å sammenligne den samme variabelen (eller uttrykket) med mange forskjellige verdier, og deretter utføre én bestemt kodesnutt avhengig av hvilken verdi det tilsvarer. Dette er nøyaktig hva `switch()`-setningen er for. Dette kan vi få til ved å bruke `if`-setningen med en rekke påfølgende `elseif`-uttrykk. Men `switch()`-setningen regnes for å være litt mer elegant. Her er et eksempel på hvordan en `switch()`-setning kan se ut:

```
<?php
switch ($i) {
    case 0:
        echo "i equals 0";
        break;
    case 1:
        echo "i equals 1";
        break;
    case 2:
        echo "i equals 2";
        break;
}
?>
```

Det er viktig å forstå hvordan `switch()`-setningen utføres for å unngå feil. Den utfører uttrykk for uttrykk. PHP fortsetter å utføre uttrykkene til slutten av `switch()`-blokken, eller første gang den ser et

break-utsagn. Hvis du ikke skriver et break-utsagn på slutten av en case, vil PHP fortsette å utføre uttrykkene fra påfølgende case. En spesiell case er default. Denne casen samsvarer med alt som ikke stemmer overens med de andre casene. For eksempel:

```php
<?php
switch ($i) {
    case 0:
        echo "i equals 0";
        break;
    case 1:
        echo "i equals 1";
        break;
    case 2:
        echo "i equals 2";
        break;
    default:
       echo "i is not equal to 0, 1 or 2";
}
?>
```

1.3 Match-kontroll

En av nyhetene i PHP8 er match-setningen. Den minner om switch, men det er likevel noen forskjeller. De skal vi komme tilbake til. Match-setningens idé er å sjekke et uttrykk mot flere alternativer. Syntaksen for match-setningen er som følger:

```
$returverdi = match(uttrykk) {
  sjekk1 => returverdi1,
  sjekk2, sjekk3 => returverdi2,

};
```

Match-setningen består, som de andre kontrollstrukturene, av et hode og en kropp. I hodet blir uttrykket vi skal sjekke plassert. Dette tilsvarer uttrykket i switch-setningen. Hver enkelt sjekk i match-setningen blir kalt en arm. Hver arm utfører en sjekk inntil en sjekk evalueres til true. Når det skjer, stopper match-setningen kjøringen og returnerer armens tilsvarende returverdi. Match-setningen vil dermed stoppe å kjøre dersom sjekk1 blir en match. Match-setningen skiller seg fra switch, ettersom switch ikke har noen returverdi. Et anenn forskjell fra switch er at match utfører en streng likhetssjekk (===) i armen mot den svakere sjekken (==) i switch. Legg merke til at match-setningen må avsluttes med et semikolon, noe som er litt uvanlig sammenliknet med andre kontrollstrukturer.

La oss se på et eksempel. Vi bruker switch-setningen i eksempel 4.10 og gjør den om til en match-setning for å sammenlikne dem. Match-setningen ser slik ut:

```php
$beskjed = match((int) date('N')) {
  1 => "I dag er det din tur å lage middag.",
  2 => "I dag er det rengjøring av rommet ditt som gjelder.",
  3 => "I dag er det din tur å gå tur med Laika.",
  4 => "Dagens oppgave er å støvsuge kjellerstuen.',
  5 => "I dag må du finne et gøy spill for hele familien :) Helg!",
  default => 'I dag er det helg og fortjent fridag. Kos deg!",
};
echo $beskjed;
```

Der switch-setningen brukte 20 linjer på samme oppgave, bruker nå match-setningen åtte linjer. Den er altså betydelig mer plassbesparende, og mer elegant må vi vel kunne si? Legg merke til at vi gjør returverdien fra date()-funksjonen om til heltall i setningens hode. Det er fordi den opprinnelig er en tekststreng, og på grunn av match-setningens krav om at ikke bare uttrykk skal matche, men også datatype, må vi omgjøre datatypen til heltall. Alternativt kan vi benytte anførselstegn på sjekkene slik at også de blir tolket som tekststrenger. Dersom match-setningen ikke oppfører seg slik du har tenkt, kan det skyldes at datatypene ikke matcher.

Denne setningen er også fleksibel ved at den kan ta flere parametere samtidig. Vi kunne for eksempel erstattet default med to parametere etter hverandre separert med et komma: 6, 7. Ved behov, kan vi plassere mange slike parametere etter hverandre. Match-setningen er også mer robust enn switch-setningen ved at den vil gi en feilmelding dersom setningen ikke matcher noen av de alternative sjekkene i armene. Dersom vi hadde utelatt *default* i eksempel 4.11, ville vi fått en feilmelding i helgedagene.

Match-setningen er på mange måter en strengere og mer moderne versjon av switch-setningen. Sistnevnte kan imidlertid sies å ha en fordel der handlingene basert på de alternative sjekkene går over flere linjer med kode.

2. Løkker

Løkker er kontroller som kjører helt til exit-kriteriet er oppfylt (noe er «usant», dvs. så lenge noe er «sant») eller i en evighet (fordi løkken har et exit-kriterium som aldri vil slå til). I de sistnevnte tilfellene vil PHP-scriptet bli stoppet av PHP-parseren etter 30 sekunder (dette er standard; du kan sjekke *max_execution_time* i php.ini om du vil se hva ditt oppsett av PHP har satt).

2.1 While-løkke

Hensikten med en `while()`-løkke er enkel. Den forteller PHP om å utføre de(n) nøstede setningen(e) gjentatte ganger, så lenge uttrykket evalueres til **TRUE**. De oppfører seg likt i PHP som i C. Verdien på uttrykket blir sjekket hver gang i begynnelsen av loopen, så selv om denne verdien endres under utførelsen av de(n) nøstede setningen(e), vil utførelsen ikke stoppe før slutten av iterasjonen (hver gang PHP kjører uttrykkene i løkken er en iterasjon). Noen ganger, hvis et uttrykk evalueres til **FALSE** helt fra begynnelsen, vil ikke de(n) nøstede setningen(e) bli kjørt én eneste gang. Her er et eksempel på en `while()`-løkke:

```
<?php
$i = 1;
while ($i <= 10) {
    echo $i++;  /* the printed value would be
                   $i before the increment
                   (post-increment) */
}
?>
```

På samme måte som med if-setningen, kan du gruppere flere utsagn sammen i en sløyfe ved å omgi en gruppe utsagn med klammeparentes. En `while()`-løkke kan gå evig, så det må du passe på. I eksemplet

ovenfor kan dette gjøres ved å la være å øke verdien på variabelen `§i` og bare skrive den ut istedenfor. Du kan prøve dette om du ønsker. Scriptet kommer til å kjøre inntil timeout.

`While()`-løkken brukes mye i PHP og har mange bruksområder. Du kan generere tabeller med den, liste opp medlemmer i et medlemssystem og hente aktiviteter fra en database bare for å nevne noen få eksempler.

2.2 For-løkker

For-løkker er de mest komplekse løkkene i PHP. De oppfører seg som sine motstykker i C. Syntaksen til en for loop er:

```
for (expr1; expr2; expr3)
      utsagn
```

Det første uttrykket (expr1) blir evaluert (utført) en gang ubetinget i begynnelsen av løkken. I begynnelsen av hver iterasjon blir expr2 evaluert. Hvis den evalueres til **TRUE**, fortsetter loopen og de(n) nestede setningen(e) utføres. Hvis den evalueres til **FALSE**, avsluttes utførelsen av løkken.

På slutten av hver iterasjon blir expr3 evaluert (utført). Hvert av uttrykkene kan være tomme eller inneholde flere uttrykk atskilt med komma. I expr2 blir alle uttrykk separert med komma evaluert, men resultatet er hentet fra den siste delen. Dersom expr2 er tom betyr det at løkken skal kjøres på ubestemt tid (PHP anser implisitt den som **TRUE**, som i C). Dette er kanskje ikke så ubrukelig som du kanskje tror, siden du ofte ønsker å avslutte sløyfen ved å bruke en betinget `break`-uttalelse i stedet for å bruke **TRUE**-uttrykket.

Se på følgende eksempler. Alle viser tallene 1 til 10:

```
<?php
/* eksempel 1 */
for ($i = 1; $i <= 10; $i++) {
    echo $i;
}

/* eksempel 2 */
for ($i = 1; ; $i++) {
    if ($i > 10) {
        break;
    }
    echo $i;
}

/* eksempel 3 */
for ($i = 1, $j = 0; $i <= 10; $j += $i, print $i, $i++);
?>
```

Bruk litt tid på de tre eksemplene og forstå hvorfor de gjør akkurat det samme. Kan du tenke deg nytteverdien av å gjøre disse oppgavene på forskjellige måter?