Institutt for informasjonssystemer

Assosiative matriser og skjemabehandling

IS-115 Webprogrammering i PHP | Peter André Busch

Innhold

Leksjon 4: matriser og skjemabehandling

I denne leksjonen skal vi gå nærmere inn på matriser og skjemabehandling. Denne leksjonen er en av de viktigste, og derfor bruker vi mye tid på den. Matriser brukes hele tiden i PHP for mange forskjellige formål. Skjema er også helt sentralt og ofte brukes matriser sammen med skjemabehandling. Derfor behandler vi disse tingene sammen. Ta deg god tid med denne modulen og forstå hva den handler om før du går videre. Kapitlene 5-6 i læreboka tar opp disse temaene og det må du lese for å komme videre.

1. Matriser

En matrise i PHP er et organisert kart. Et kart knytter verdier til nøkler. Denne typen er optimalisert for flere forskjellige bruksområder; det kan behandles som en matrise, liste (vektor), hashtabell (en implementering av et kart), ordbok, samling, stabel, kø og sannsynligvis mer. Siden matriseverdier også kan være andre matriser, er trær og flerdimensjonale matriser også mulige.

1.1 Opprette en matrise

For å opprette en matrise brukes den innebygde funksjonen `array()`. Syntaksen "indeks => verdier", atskilt med komma, definerer indeksen og verdiene. Indeksen kan være av typen streng eller heltall. Når indeksen er utelatt, genereres det automatisk en indeks av et heltall (begynner fra 0). Hvis indeksen er et heltall, vil den neste genererte indeksen være den største heltallindeksen + 1. Merk at når to identiske indekser er definert, vil sistnevnte overskrive førstnevnte. Her er noen eksempler:

```php
<?php
$array = array(1, 2, 3, 4, 5, 8 => 1,  4 => 1, 19, 3 => 13);
print_r($array);
?>
```

Test gjerne ut dette scriptet for å se hva som skrives ut på skjermen. Skjønner du hvorfor det blir slik? Prøv dette ut før du ser på forklaringen nedenfor.

Forklaringen er dette. 1-tallet først har ingen indeks definert, derfor genereres indeks automatisk så den blir null (mao. så vil du finne 1-tallet på plass 0 i matrisen). 2-tallet har heller ingen indeks og derfor genereres ny indeks som blir 0+1 (mao. så vil du finne 2-tallet på plass 1 i matrisen). Lengre ut kan du imidlertid se at både indeks og verdi defineres (`8 => 1`) som betyr at et 1-tall blir lagt på plass nr. 8 i matrisen. Det neste som skjer er at et 1-tall blir lagt på plass nr. 4 i matrisen (her var det fra før et 5-tall, slik at det overskrives). Indeks for 19-tallet genereres automatisk fordi indeks ikke er definert (dermed blir indeks 8+1 fordi 8 er høyeste indeks til nå – matrisen fyller ikke opp ev. tomme plasser). Til slutt legges verdien 13 inn på plass 3 (her var det fra før et 4-tall som blir overskrevet).

En annen mulighet er å definere en streng som indeks, da kan det se slik ut:

```php
<?php
$foo = array('bar' => 'baz');
echo "Hello " . $foo['bar'] . "!"; // Hello baz!
echo "Hello {$foo['bar']}!"; // Hello baz!
?>
```

Variabelen `$foo` settes til en matrise med indeks 'bar' som har verdien 'baz'. Du kan se hvordan du skriver ut verdiene i matrisen i de to `echo`-setningene under. Begge gjør det samme, mens nr. 2 gjør det

inni en tekststreng derfor kreves klammeparentes rundt. Det går også an å ha flerdimensjonale matriser, men for dette henviser jeg til php.net, så kan dere utforske litt selv med å teste ulike script.

1.2 Endre en matrise

Det er veldig enkelt å endre verdiene i en matrise. Det har vi allerede gjort i forrige avsnitt og det gjøres ved å overskrive matrisen. For eksempel:

```
<?php
$foo = array('bar' => 'baz');
$foo['bar'] = 'pox';
?>
```

Nå vil verdien av matrisen `$foo` med indeks `bar` ha verdien `pox`. Det samme gjelder matriser med heltall som indeks. Om matrisen har flere indekser, så blir kun den ene plassen med den angitte indeks overskrevet.

1.3 Slette en matrise

Du kan enkelt slette en matrise (f.eks. `$foo`) ved å bruke funksjonen `unset`. Det ser noe slikt ut:

```
<?php
unset($foo);
?>
```

Hvis du forsøker å kalle matrisen `$foo` vil den ikke lengre eksistere. Du kan også bruke denne funksjonen til å slette enkelte elementer fra matrisen.

1.4 Manipulering av matriser samt tips og triks

Det finnes en rekke innebygde funksjoner for å manipulere matriser. Du finner alle disse på **php.net** (se **https://www.php.net/manual/en/ref.array.php**). Her ser du at det er en lang rekke med funksjoner som du kan utforske selv. Du vil finne ut at dette nettstedet vil bli brukt en del for å finne ut hvordan du skal løse ulike problemstillinger. En av funksjonene som vi kan bruke med matriser presenterte vi i forrige avsnitt (`unset`). Her skal vi se på noen andre funksjoner som kan hjelpe oss å oppnå ulike ting. Den første vi skal se på er funksjonen `array_pop`:

```
<?php
$stack = array("orange", "banana", "apple", "raspberry");
$fruit = array_pop($stack);
print_r($stack);
?>
```

Denne funksjonen «popper» av det siste elementet i matrisen. Det betyr at matrisen inneholder verdiene `"orange", "banana", "apple"` etter at `array_pop()` er utført og at `$fruit` inneholder `"raspberry"`. La oss anta at matrisen inneholder tall istedenfor og vi ønsker å sortere matrisen etter tallverdi slik at det høyeste tallet kommer først og det minste tallet kommer til slutt. Da kan vi gjøre noe slikt som:

```
<?php
$comp = array("Jacob" => 5, "Silje" => 17, "Andrea" => 1, "Ole" => 14);
arsort($comp );
```

```
foreach ($comp as $contestant => $points) {
    echo "$ contestant = $ points\n";
}
?>
```

Det som skjer er at matrisen `$comp` sorteres i reversert rekkefølge (`asort()` sorterer motsatt vei) slik at de høyeste verdiene kommer først og de laveste til slutt. Da blir resultatet noe slikt som:

```
Silje = 17
Ole = 14
Jacob = 5
Andrea = 1
```

Legg merke til at indeksen til verdien følger med i sorteringen (så selv om `"Jacob"` okkuperte den første plassen i matrisen innledningsvis, så er gjør han ikke det etter sorteringen). Om du vil rote til matrisen etterpå kan du bruke `shuffle()`. Om du ønsker å sortere på indeks istedenfor verdi bruker du funksjonene `ksort()` og `krsort()`. Et annet eksempel er `count()`. Denne teller hvor «stort» matrisen er. I ovennevnte eksempel ville svaret blitt 4. Enda et nyttig eksempel er funksjonen `in_array()`. Denne sjekker om en verdi eksisterer i matrisen. Det kan f.eks. se slik ut:

```
<?php
$os = array("Mac", "NT", "Irix", "Linux");
if (in_array("Irix", $os)) {
    echo "Got Irix";
}
if (in_array("mac", $os)) {
    echo "Got mac";
}
?>
```

Her ser dere at verdiene som det søkes etter finnes i matrisen. Fordi `in_array()`-funksjonen returnerer **TRUE** eller **FALSE**, så kan den sjekkes direkte med en if-setning. Om du ønsker å sjekke om en indeks eksisterer, så kan du bruke `array_key_exists()`. Til slutt ønsker jeg å nevne funksjonen `array_diff()` som sammenlikner to matriser og skriver ut forskjellen mellom dem. Her må du huske at rekkefølgen på matrisene når funksjonen kalles er viktig. Som allerede sagt, så er det mange flere funksjoner som du kan kikke på under oppføringen på php.net.

1.5 Anvendelsesområder

Her skal vi presentere noen eksempler på hvordan assosiative matriser benyttes i PHP. Det som gjør dem så anvendbare er at de fungerer som midlertidig lagringsplass for ulik type informasjon. De kan sorteres og organiseres nokså lett og det er enkelt å hente ut data i matrisene. Dette gjør at de kan brukes til å lagre informasjon fra et skjema inntil denne informasjonen blir lagret i en database, holde styr på poeng i en konkurranse eller for å organisere data som blir hentet fra en database. Dette er bare noen få eksempler. Det finnes mange andre anvendelsesområder som vil føre for vidt å beskrive her. I neste avsnitt skal vi bruke matriser for å behandle informasjon i et skjema.

2. Skjemabehandling

Skjemaer er helt sentrale i webutvikling. De brukes som kontaktskjema, for å lagre informasjon og noe så elementært som å logge seg på en tjeneste. Uten skjemaer vil det bli vanskelig å skape interaktivitet mellom nettstedet og brukeren. Du kan jo selv bare tenke på hvor mange ganger du bruker et skjema på web.

2.1 Opprette et skjema

Å opprette et skjema er en tosidig oppgave. Den ene siden handler om å lage til HTML- og CSS-kode slik at skjemaet blir tilgjengelig for brukeren. Dette er en relativt enkelt oppgave (selv om det kan bli noe plunder med spesielt CSS-koden). Det er utenfor hensikten med dette emnet å bruke tid på CSS, så det må du se på selv. Vi skal fokusere på den programmeringstekniske siden av skjemahåndteringen.

Hvordan vet jeg at skjemaet er sendt?

All informasjon som sendes gjennom et skjema blir gjort tilgjengelig gjennom matrisen `$_POST`. Dermed kan vi bruke denne matrisen for å sjekke om skjemaet er sendt. Begynnelsen av skjemaet kan se slik ut:

```
<form name="frmNyhet" action="<?php echo $_SERVER['PHP_SELF'] . '?id=' . $_GET['id'] .
'&aid=' . $_GET['aid']; ?>" method="post" enctype="multipart/form-data"
class="orderForm">
```

Verdien i feltet *action* viser hvor dataene skal sendes etter at skjemaet er fylt ut og sendt. I dette tilfellet er det samme fil som skal utføre handlingene. Jeg skal forklare `$_GET` litt senere. Metoden her er post hvilket betyr at dataene blir tilgjengelige gjennom matrisen `$_POST`. Dette er mest vanlig og det anbefales. Hva det innebærer kommer vi tilbake til. HTML-koden som viser en lagre-knapp i et skjema kan se slik ut:

```
<td class="buttBox" colspan="2"><input type="submit" value="Lagre"
name="frmNyhetLagre" /></td>
```

Det sentrale du skal fokusere på akkurat her er det som står i feltene 'name' og 'value'. Det første feltet blir indeks i `$_POST`-variabelen mens den sistnevnte blir verdien som settes. Med andre ord, dersom du vil sjekke om et skjema er sendt, så sjekker du om `$_POST['frmNyhetLagre']` har verdien 'Lagre' (egentlig holder det å sjekke om den har en verdi overhodet). Dette kan du sjekke med en if-setning og dersom skjemaet ikke er sendt, så ignoreres skjemabehandlingen.

Dersom du har spesiell informasjon som skal skjules fra de som kan se på skjermen (f.eks. passord), så definerer du et felt som passord på denne måten: `<input type="password" name="passord" />`.

Hvordan kan jeg hente informasjonen fra skjemaet?

Informasjonen fra resten av skjemaet håndteres også ved hjelp av matrisen `$_POST`. Indeks er det samme som verdien oppgitt i name-feltet i HTML-koden (se eksemplene ovenfor og nedenfor).

```
<td class="value"><input type="text" size="30" name="frmNyhetTittel"
id="frmNyhetTittel" value="<?php echo $_POST['frmNyhetTittel']; ?>" /></td>
```

Det vil si at om du har et felt som skal hente gateadresse (og du bruker dette som verdi i name-feltet), kan du hente gateadressen som brukeren har fylt ut i skjemaet gjennom variabelen `$_POST['gateadresse']`.

Hva kan jeg gjøre med informasjonen fra skjemaet?

Et skjema har flere bruksområder og her er noen av dem. Når et skjema er sendt er det mest vanlige å lagre dataene i en database. Hvordan du kan lagre informasjon i en database skal vi gå gjennom senere. Når du har lært det, er det relativt enkelt å lagre informasjonen der. En annen mulighet er å sjekke informasjonen med en database (f.eks. for kredittkortinformasjon) eller en annen form for API. En tredje mulighet for å søke etter informasjon (f.eks. i en database). Og til slutt er det en mulighet å bruke informasjonen i skjemaet til å sende e-post. Dette er det som gjøres med kontaktskjema. Hvordan du kan sende e-post ved hjelp av PHP skal du lære i neste modul. Uansett bruksområde er det viktig å sjekke hvilken informasjon brukeren faktisk har oppgitt i skjemaet. Det kan nemlig være at brukeren har slurvet, skrevet feil og/eller bevisst forsøker å skrive inn skadelig kode i skjemaet. I beste fall går det utover kvaliteten på dataene i databasen – i verste fall kan data blir slettet eller manipulert. Nå skal vi se på hvordan skjemaer kan sjekkes.

Forskjellen på POST- og GET-metodene

Som innledningsvis nevnt, så er det også mulig å bruke `$_GET`. Du finner en drøfting av disse i avsnittene 4.2.2 - 4.2.4. Forskjellen er at ved bruk av GET vil informasjonen i skjemaet blir sendt som en del av URLen. Dermed kan du i adresselinjen se mye informasjon og det kan fort bli noe uryddig. Derfor foretrekkes ofte `$_POST` i disse tilfellene. `$_POST` sender informasjon uten at du ser den (den er ikke kryptert eller hemmelig av den grunn). Dersom du skal sende passord er det en stor fordel å ikke bruke `$_GET`. Du kan også manipulere informasjonen i URLen ved bruk av `$_GET`. `$_GET`-metoden brukes mye for å angi at man ønsker informasjon om et bestemt medlem eller for å laste én bestemt nyhetsartikkel. F.eks. ved å legge til id: medlem.php?id=1. Her kan du hente verdien i feltet id med å bruke `$_GET['id']` og dermed vise medlemsprofilen for medlem med f.eks. medlemsnummer 1 (eller bare unik ID i databasen lik 1). Dersom du har behov for flere felt som du kan hente informasjon fra, så bruker du &-tegnet: medlem.php?id=1&hash=jdhYTdbv.

2.2 Sjekk av skjema

Ofte er det nødvendig med en kontrollsjekk av skjemaet før informasjonen f.eks. lagres i en database. Typiske sjekker vil være å sjekke om alle obligatoriske felt er fylt ut og om enkelte felt inneholder «ulovlige» verdier (f.eks. tallverdier i navnefeltet eller kode). Av hensyn til integriteten til databasen er det viktig at denne jobben gjøres. Det kan også være sikkerhetsmessige hensyn å ta. Å sjekke et skjema innebærer derfor at skjemahåndteringen blir litt mer kompleks fordi sjekken må utføres og ev. feilmeldinger må kommuniseres til brukeren.

Det kan være lurt å lage egne stilregler i CSS for feilmeldinger slik at de vises likt gjennom hele nettstedet samt at feilmeldingene blir tydelige for brukeren. Dette kan gjøres ved å vise en feilmelding eller markere hvilke felt i skjemaet som behøver oppmerksomhet. Dette er på mange måter en smakssak så lenge det gjøres tydelig for brukeren.

Det kan være en god idé å sjekke flere ting i skjemaet og etter hvert som feil dukker opp blir disse lagret i en matrise som deretter kan sjekkes. Dersom matrisen ikke eksisterer, betyr det at ingen feil oppsto underveis. Dersom matrisen eksisterer, så kan man iterere gjennom det og vise innholdet – og dermed også avvente med å lagre informasjonen i databasen. Her er et eksempel på en feilsjekk:

```php
<?php
if ($_POST['formSubmitted'])
{
    // Creates empty array that can be filled with error messages if needed
    $errorMsg = array();
```

```php
        // Checks if first name is filled out in form
        if (!$_POST['formFName'])
        {
             // Not filled out - adds error message to array
             $errorMsg['A1'] = 'Fornavn må oppgis';
        }

        // Checks if last name is filled out in form
        if (!$_POST['formLName'])
        {
             // Not filled out - adds error message to array
             $errorMsg['A2'] = 'Etternavn må oppgis';
        }

        // If array is empty; no error messages have been triggered
        if (!count($errorMsg))
        {
            // Save to database
        }
        else
        {
           // Sort array
           ksort($errorMsg);

           // Iterate through array and display error messages
           foreach ($errorMsg as $key => $val) {
               echo "$key" . ": $val\n";
           }
        }
    }
    ?>
```

I dette skjemaet gjøres to sjekker som begge handler om obligatoriske felt som ikke er utfylt. Indeks i matrisen er laget for å ev. gruppere feilmeldingene logisk sammen (det som har med navn burde komme samtidig). Derfor blir matrisen sortert til slutt etter indeks. Du bør i tillegg til disse sjekkene sørge for at det ikke er noe galt i navneformatet (det kan være komplisert basert på hva du krever) – det minste som bør gjøres er å sørge for at navn lagres med stor forbokstav og resten med små bokstaver. Av sikkerhetsmessige grunner bør også ev. kode alltid fjernes fra utfylte felt.