

**Systemek**<sup>®</sup>

-muliggjør en mobil og sikker hverdag



# Introduksjon til PHP

**KOMPIS - 11. august 2009**

**Ivar Breivik, Systemek AS**

**[breivik@systemek.no](mailto:breivik@systemek.no)**



## Agenda

PHP er et server-side scriptspråk som er spesielt egnet for raskt å lage dynamisk genererte web-sider. I dette foredraget vil vi få en lett introduksjon til språket som benyttes på 1/3 av alle web-servere på Internett.

### Agenda:

- litt historie og gjennomgang av den viktigste syntaksen
- utviklingsverktøy og installasjon sammen med Apache og MySQL
- demo: lagre til database og bruk av GD bibliotek

### Foredraget er for utviklere som ønsker:

- en innføring i PHP som scriptspråk
- å sette opp en enkel web-applikasjon med PHP, Apache og MySQL

### Krav til deltaker:

- grunnleggende programmering
- kjennskap til web-applikasjoner og HTML



# Hva er PHP?

## PHP ...

- er primært et server-side scriptspråk
- er et scriptspråk med syntaksiske likheter til C, Java og Perl
- er spesielt egnet for web-utvikling
- er lett å lære for utviklere
- er velbrukt og velprøvd
- er Open Source, gratis å laste ned og bruke
- kjører på ulike plattformer (Windows, Linux, Unix)
- har støtte for de fleste web-servere (Apache, IIS, ..)



## Hva er PHP?

Målet med PHP er å la web-utviklere lage dynamisk genererte web-sider raskt.

De aller fleste PHP prosjekter/installasjoner:

- lite omfang
- lage enkle nettsteder med enkel navigering
- bruke eksisterende rammeverk
- etv. utvikle plugins til disse

Men det finnes selvsagt store som bruker PHP også:

- Facebook, Yahoo, Wikipedia, Digg, YouTube

PHP står for PHP: Hypertext Preprocessor



## Historien bak PHP

Opprinnelsen til PHP ble til i 1995, da Rasmus Lerdorf lagde "Personal Home Page Tools" i Perl for å holde styr på hvor mange som besøkte hjemmesiden han. Etterhvert utvidet Rasmus funksjonaliteten med å tilby håndtering av variabler fra HTML-forms og databasetilgang.

Og allerede i 1997, etter en omskriving til C og blitt åpen kildekode, hadde PHP blitt installert på over 50 000 nettsted, som da tilsvarte rundt 1% av alle domener den gangen.

PHP 3.0 ble utgitt i 1998 og lignet mye på slik PHP er i dag. På topp var denne versjonen installert på omtrent 10% av alle web-servere på Internett.



## Historien bak PHP

PHP 4.0 kom i 2000 og var en total omskriving med inkludering av Zend Engine og støtte for mange flere web-servere, HTTP-sesjoner, buffering, sikker håndtering av bruker-input, 3. parts databaser og API'er. Antall utviklere som brukte PHP var over 100 000 og PHP-teamet bestod av over 50 utviklere. Flere millioner nettsteder har tatt i bruk PHP og dette utgjør ca. 20% av alle domener.

PHP 5.0, med Zend Engine 2.0, kom i 2004 og har ny støtte for objektorientert programmering og en mengde nye funksjoner.

PHP var ved inngangen til 2009 installert på 33% av alle web-servere på Internett. ASP: 21%, skjult: 45% (kilde: nexen.net).

PHP 5.3 er siste versjon fra 30. juni 2009. Arbeid med PHP 6.0 pågår.



## Erfaring: Negativt med PHP

- Mengden funksjoner er enormt og det kan være noe vanskelig å finne frem til akkurat den man trenger.
- Av og til opplever man å skrive en blokk med generell kode, bare for å finne ut at det finnes en funksjon som gjør akkurat det.
- Det er veldig lett å blande datamodell, visning og forretningslogikk, og ende opp med spaghetti-kode.
- Moduler som er kompilert inn varierer fra webhotell til webhotell (spesielt XML parsing og GD grafikkbiblioteker), så man må tenke installasjon og drifting mens man utvikler.



## Erfaring: PHP på større nettsted

Rapport: Experiences of Using PHP in Large Websites

Fokus: valget av PHP gjør utvikling av dynamiske websider raskt og enkelt, men hvordan skalerer PHP for store kommersielle nettsteder?

- dårlig separasjon mellom presentasjon og forretningslogikk
- hovedsaklig fokus på "enslige" utviklere, team med utviklere går veldig lett i beina på hverandre
- fallgruver når man deployer er problemer som er vanskelige å finne i utvikling
- overforenkling fører til enda mer kompliserte løsninger
- fungerer godt for uerfarne utviklere som går fra rent HTML-design til utvikling, dog for mer erfarne vil enkelheten raskt føre til med kompleksitet og sakke ned utviklingsprosessen
- mangel på støtte for arbeid på store prosjekter



## Hvordan kode PHP?

PHP kodes i vanlige tekstfiler som legges på web-serveren for kjøring. Kan også kjøres standalone uten web-server.

- Filene kan inneholde tekst, HTML og/eller PHP-kode
- Filene returneres til browseren som ren tekst/HTML
- Filene har filnavn som ender på .php
- PHP-kode må omslutes av tagger for start (`<?php` eller `<?`) og slutt (`?>`) på tilsvarende måte som i JSP- og ASP-filer
- Alle kodelinjer må avsluttes med semikolon
- Kommentarer skrives på samme måte som i Java og C:  
`//` for en linjes kommentar og `/*` og `*/` for flere linjer



# Hvordan kode PHP?

Eksempel index.php:

```
<?php  
    kode;  
    // kommentar  
?>
```

HTML, tekst

```
<?php  
    mer kode;  
?>
```

HTML <?php kode;?> HTML



# Syntaks flytkontroll

Vanlige instruksjoner for tilstandssjekk og løkker finnes:

```
if (test1) {  
    // blir utført hvis test1 er true  
} elseif (test2) {  
    // blir utført hvis test2 er true  
} else {  
    // blir utført hvis test1 og test2  
    // er false  
}
```

```
switch (n) {  
    case verdi1:  
        // blir utført hvis n == verdi1  
        break;  
    case verdi2:  
        // blir utført hvis n == verdi2  
        break;  
    default:  
        // blir utført hvis n er ulik  
        // både verdi1 og verdi2  
}
```

```
while (test1) {  
    // blir utført så lenge  
    // test1 er true  
}
```

```
do {  
    // blir utført så lenge  
    // test1 er true  
} while (test1);
```

```
for (init; test1; inkrement) {  
    // blir utført så lenge  
    // test1 er true  
}
```

samt foreach, include,  
require, return og goto



# Variabler

## Variabler i PHP:

- starter med dollar-tegn (\$)
- navn må starte med en bokstav og kan bare inneholde alfanumeriske tegn og understrek (a-z, A-Z, 0-9 og \_)
- blir automatisk deklarerert hver gang de får tildelt en verdi
- PHP tildeler variabelen korrekt type, basert på verdien den har

```
$tekst = "Tekst"; // string
```

```
$heltall = 4; // int
```

```
$desimal = 3.14; // float
```



## Variabler - boolean

Det finnes fire primitive variabeltyper i PHP:

- boolean
  - TRUE/FALSE (case-insensitive)
  - andre typer blir automatisk konvertert til boolean hvis operatoren, funksjonen eller kontrollstrukturen krever et boolsk uttrykk:
    - FALSE = tallet 0, 0.0, tom streng, strengen "0", et tomt array, ...

```
$test = TRUE;  
if ($test) { ... }  
$tekst = "";  
if ($tekst) { ... }
```



# Variabler - integer

- integer
  - heltall representert i base desimal, hexadecimal (0x) eller oktalt (starter med 0)
  - PHP støtter ikke unsigned integer
  - størrelsen er plattform-avhengig, men er som regel 32 bit (bruk PHP\_INT\_SIZE for å sjekke)
  - bruk GNU Multiple Precision (GMP) bibliotek ved behov for vilkårlig lengde på heltall
  - ved overflyt blir variabelen konvertert til float

```
$a = 1234; // decimal number  
$a = -123; // a negative number  
$a = 0123; // octal number (equivalent to 83 decimal)  
$a = 0x1A; // hexadecimal number (equivalent to 26 decimal)
```



# Variabler - float

- float
  - reelt tall representert av en heltallsdel og en fraksjonsdel skilt med desimalskilletegn (punktum) og/eller en eksponentdel
  - størrelsen er plattform-avhengig, men er som regel 64 bit IEEE format (1.8e308 med en presisjon på 14 desimaltegn)
  - ved behov for større presisjon eller vilkårlig lengde, bruk GMP eller BC Math bibliotek
  - begrenset nøyaktighet gjør at tallene sjelden blir representert helt korrekt
    - unngå sammenlikne likhet mellom float
    - ikke bruk float til pengebeløp og andre verdier som krever eksakte resultater

```
$a = 1.234;
```

```
$b = 1.2e3; // 1200
```

```
$c = 7E-10; // 0.0000000007
```



## Variabler - string

- string
  - en serie med tegn omsluttet av ' eller ''
  - ' er strengere mht. escaping av spesialtegn
  - '' tilbyr ekspandering av variabler inne i strengen
  - 1 tegn = 1 byte (ingen innebygd støtte for Unicode < PHP 6)
  - ''ingen'' begrensning på lengden
  - sammenslåing av strenger gjøres med punktum ( . )

```
echo 'Linje 1\nLinje 2';  
Linje 1\nLinje 2
```

```
echo "Linje 1\nLinje 2";  
Linje 1  
Linje 2
```



## Variabler - string

- ekspandering og sammenslåing av strenger med dått:

```
$by = "Oslo";
```

```
$innbyggere = 578870;
```

```
$tekst = "$by har $innbyggere innbyggere";
```

```
$tekst = $by . " har " . $innbyggere . " innbyggere";
```

- finnes mengder av funksjoner for behandling av strenger:
  - standard strengbibliotek > 80 funksjoner
  - regular expression (søk, erstatt, splitt) > 20 funksjoner
  - behandling av URL'er > 10 funksjoner
  - kryptering, hash, sikkerhet > 40 funksjoner
  - XML parsing > 10 bibliotek



## Variabler

I tillegg til de primitive variablene finnes også to sammensatte:

- array
- object

I tillegg finnes noen spesialtyper:

- resource - filer, databasetilkoblinger og andre ressurser
- NULL - variabler som ikke har fått noen verdi enda, har blitt satt til NULL eller kalt med unset()
- mixed - brukes i manualen for å indikere at et parameter til en funksjon takler flere typer
- callback - funksjoner som parametre
- void



# Variabler - arrays

Arrays er en viktig og fleksibel del av PHP:

- brukerinntak og aksess mot datalag (database) bruker ofte arrays for å utveksle data
- internt er arrays i PHP implementert som et "ordered map" som er optimalisert for mange ulike bruk, blant annet som array, list (vector), hash table, dictionary, collection, stack, queue med flere.

Det finnes tre typer arrays i PHP:

- numeriske array (der indeksen er et tall)
- assosiative array (der indeksen er assosiert med en strengverdi)
- multidimensjonale array (et array som inneholder et eller flere arrays)



## Variabler - arrays

- Numeriske array
  - Oppretting vha. indeks manuelt:

```
$byer[0] = "Oslo";  
$byer[1] = "Bergen";  
$byer[2] = "Trondheim";
```
  - Oppretting vha. array-funksjonen:

```
$byer = array("Oslo", "Bergen", "Trondheim");
```
  - Oppretting vha. indeks automatisk:

```
$byer[] = "Oslo";  
$byer[] = "Bergen";  
$byer[] = "Trondheim";
```
  - Oppretting vha. array\_push():

```
array_push($byer, "Oslo");  
array_push($byer, "Bergen");  
array_push($byer, "Trondheim");
```



# Variabler - arrays

- Bruk av numeriske array:

```
echo $byer[1] . " er den fineste byen i Norge";
```
- Bruk av indeks som går ut over lengden på arrayet, gir NULL tilbake (tom streng ved utskrift), altså ingen feilmelding tilsvarende `ArrayOutOfBoundsException`
- Slette en verdi fra array:

```
unset($byer[2]); // Fjerner Trondheim
```
- I tillegg til vanlig for- og while-løkke er det en egen løkkeinstruksjon for arrays:

```
foreach ($array as $verdi) {  
    // Koden blir utført for hver verdi i $array og  
    // $verdi blir tildelt neste verdi i $array  
}
```



## Variabler - arrays

- Assosiative array (tilsvarende hashmap)

- Oppretting manuelt:

```
$byer_grunnlagt["Oslo"] = 1048;  
$byer_grunnlagt["Bergen"] = 1070;  
$byer_grunnlagt["Trondheim"] = 997;
```

- Oppretting vha. array-funksjonen:

```
$byer_grunnlagt = array("Oslo" => 1048,  
                        "Bergen" => 1070,  
                        "Trondheim" => 997);
```

- Bruk av assosiative array:

```
$by = "Bergen";  
echo $by . " ble grunnlagt i " . $byer_grunnlagt[$by];
```



## Variabler - arrays

- Løkkeinstruksjon for assosiative arrays:

```
foreach ($array as $indeks => $verdi) {  
    // Koden blir utført for hver verdi i $array,  
    // $verdi blir tildelt neste verdi og  
    // $indeks blir tildelt neste indeks  
}
```

```
foreach ($byer_grunnlagt as $by => $grunnlagt) {  
    echo $by . " ble grunnlagt i " . $grunnlagt;  
}
```

- Sorteringsfunksjoner:

```
sort($byer_grunnlagt); // sorterer på verdi: årstall  
ksort($byer_grunnlagt); // sorterer på indeks/nøkkel: bynavn
```



# Variabler - arrays

- Multidimensjonale array

```
$byer_info = array("Oslo" => array(1048, 578870),  
                  "Bergen" => array(1070, 252918),  
                  "Trondheim" => array(997, 168988));
```

```
$by = "Oslo";
```

```
echo $by . " ble grunnlagt i " . $byer_info[$by][0] . " og har "  
      . $byer_info[$by][1] . " innbyggere";
```

```
$byer_info = array("Oslo" => array("Grunnlagt" => 1048,  
                                  "Innbyggere" => 578870),  
                  "Bergen" => array("Grunnlagt" => 1070,  
                                  "Innbyggere" => 252918),  
                  "Trondheim" => array("Grunnlagt" => 997,  
                                       "Innbyggere" => 168988));
```

```
$by = "Trondheim";
```

```
echo $by . " ble grunnlagt i " . $byer_info[$by]["Grunnlagt"] . "  
      og har " . $byer_info[$by]["Innbyggere"] . " innbyggere";
```



# Variabler - arrays

Verdiene i et array i PHP er også løst koblet, de trenger ikke å være av samme type:

```
$byer_grunnlagt["Oslo"] = 1048;  
$byer_grunnlagt["Bergen"] = array(1050, 1070);  
$byer_grunnlagt["Trondheim"] = "Rundt 1000";
```

Sortering med brukerdefinert funksjon:

```
usort($byer_grunnlagt, "sortering_grunnlagt");
```

```
function sortering_grunnlagt($a, $b) {  
    return 0; // hvis like  
    return 1; // hvis $a > $b  
    return -1; // hvis $a < $b  
}
```

Debugtips:

```
print_r($array);
```



## Variabler - objects

PHP tilbyr objektorientering programmering tilsvarende Java.

- Alle klasser må deklarereres før bruk, men det er ingen sammenheng mellom filnavn og klassenavn
- Autoloading - laster klasser automatisk
- Inherit
- Constructor/destructors
- Visibility
- Abstraction
- Interfaces
- Overloading
- Object cloning and comparing
- Reflection



# Variabler - objects

```
class Sted {
    // medlemsvariabler
    public $navn = "ukjent sted";

    // konstruktør og metodedeklarasjoner
    public function __construct($navn) {
        $this->navn = $navn;
    }
    public function navn() {
        return $this->navn;
    }
}

$sted = new Sted("Fornebu");
echo $sted->navn;    // variabel: Fornebu
echo $sted->navn(); // metode: Fornebu
echo Sted::navn;    // statisk variabel: ukjent sted
echo Sted::navn(); // statisk metode: krasjer pga. $this
```



## Variabler - dato / tid

Det finnes ingen egen primitiv type for dato / tid i PHP, UNIX timestamp blir som regel brukt, eller evt. DateTime-biblioteket (DateTime og Calendar).

Formatering av datoer:

```
date(format); // returnerer dagens dato/tid formatert
```

```
$idag = date("d.m.Y"); // "11.08.2009"
```

Datomanipulasjon med mktime():

```
mktime(hour, minute, second, month, day, year);
```

```
// returnerer et UNIX timestamp (antall sekunder fra 1.1.1970)
```

```
// støtter overflyt av parametre
```

```
$timestamp = mktime(0, 0, 0, date("m"), date("d") + 1, date("Y"));
```

```
$imorgen = date("d.m.Y", $timestamp); // 12.08.2009
```



## Operatorer

Alle de vanlige operatorene finnes også i PHP:

- Aritmetiske: +, -, \*, /, %, ++ og --
- Tildeling: =, +=, -=, \*=, /=, .= og %=
- Sammenlikning: ==, !=, >, <, >= og <=
- Logiske: &&, || og !
- .= brukes ofte ved "bufring" av tekst før den sendes til output:

```
$buffer .= "html kode";  
$buffer .= "mer html kode";  
// behandle $buffer  
// før den skrives ut  
echo $buffer;
```



## Sjonglering med variabler

I og med at PHP ikke krever eksplisitt typedefinisjon ved deklarasjon av variabler, er det spesielle regler for automatisk typekonvertering ved bruk av operatorer.

```
$a = "0"; // $a er string
```

```
$a += 2; // $a er integer 2
```

```
$a = $a + 1.3; // $a er float 3.3
```

```
$a = 5 + "10 epler"; // $a er integer 15
```

```
$a = 5 + "eple"; // $a er integer 5
```

```
$a = "2.5 eple" + 2; // $a er float 4.5
```



## Funksjoner

PHP kommer innebygd med over 700 funksjoner og språket er veldig funksjonssentrert, spesielt før støtten for objektorientert programmering kom.

Bruk tid til å studere utvalget av funksjoner for å unngå å lage noe noen har laget før:

- <http://no.php.net/manual/en/funcref.php>



## Funksjoner

Egendefinerte funksjoner er måten man samler generell logikk i applikasjonen sin på:

- funksjonsnavn følger samme regler som for variabler (alfanumeriske tegn)
- all gyldig PHP-kode kan brukes i en funksjon (også andre funksjoner og klassedefinisjoner)
- både variabelt antall parametre og default parametre støttes
- funksjoner må deklarereres før de kalles hvis ikke de deklarereres i samme fil som de blir brukt
- funksjoner blir tilgjengelig i "global space"



# Funksjoner

```
function funksjonsnavn($param1, $param2 = "default") {  
    // kode som skal utføres  
    // kan bruke func_get_args() for å aksessere input-variablene  
    return $returverdi;  
}  
  
$resultat = funksjonsnavn("test");  
$resultat = funksjonsnavn("test", "annen verdi", 123);
```



## Funksjoner - pass-by-value

Alle parametre blir overført som pass-by-value, slik at selv om parameteren blir endret inne i funksjonen, beholder den sin originale verdi utenfor. Dette gjelder også arrays og innholdet i dem.

```
function endre_innhold($array) {  
    $array[0] = 999;  
}
```

```
$array = array(123, 234, 345);  
endre_innhold($array);  
print_r($array);
```

```
123, 234, 345
```



## Funksjoner - pass-by-reference

For å kunne endre verdien til en parameter slik at den endres også utenfor funksjonen når den returnerer, brukes pass-by-reference med &:

```
function endre_innhold(&$array) {  
    $array[0] = 999;  
}
```

```
$array = array(123, 234, 345);  
endre_innhold($array);  
print_r($array);
```

```
999, 234, 345
```



# Funksjoner - retur av endret parameter

En annen måte å gi tilgang til det endrede arrayet er å returnere det. Praktisk hvis man skal bruke det videre i en annen funksjon, men vil ta vare på det originale:

```
function endre_innhold($array) {  
    $array[0] = 999;  
    return $array;  
}
```

```
$array = array(123, 234, 345);  
$endret_array = endre_innhold($array);  
print_r($endret_array);
```

999, 234, 345



## Server Side Include

Når web-applikasjonen vokser seg ut over en side og man ønsker litt mer gjennomførte og uniforme sider, samler man felles kode og visning i egne filer og inkluderer dem i applikasjonen.

Dette gjøres med `include()` eller `require()`, som begge inkluderer en PHP-fil inn i en annen før serveren utfører dem. Sistnevnte feiler og stopper applikasjonen hvis filen ikke finnes.

```
include("header.php");  
include("menu.php");  
include("body.php");  
include("footer.php");
```

Hvis da linker til en side endres, er det bare nødvendig å gjøre endringen i `menu.php` og ikke alle sider.



# Håndtering av forms (bruker-input)

Bruker-input fra HTML-forms gjøres automatisk tilgjengelig via noen spesielle globale array-variabler.

- **\$\_GET**
  - brukes for å hente verdier fra vanlige URL'er og HTML forms med method="get" (med de begrensningene det medfører: maks. lengde på URL, passord osv. vises direkte i URL)
- **\$\_POST**
  - brukes for å hente verdier fra HTML forms med method="post"
- **\$\_REQUEST**
  - inneholder verdier fra både \$\_GET og \$\_POST (samt cookies), scriptet bryr seg ikke om det er en GET eller POST



# Håndtering av forms (validering)

Noen råd om validering av bruker-input:

- Er verdien satt?
  - `isset($_REQUEST["xxx"])` for å sjekke om xxx er satt
- Tall:
  - `is_numeric($_REQUEST["xxx"])` for å sjekke om xxx er et tall
  - `$tall = (int) $_REQUEST["yyy"]; // blir satt til tallet 0 hvis yyy inneholder noe annet enn tall`
- Escaping av strenger:
  - Strenger som inneholder `\ ' "` og en del andre tegn blir escaped med `\`. Bruk `stripslashes($tekst)` for å unescape hvis det er nødvendig.



# Håndtering av forms (eksempel)

endre\_by.html:

```
<form action="endre_by.php" method="post">
  By: <input type="text" name="by"/>
  Grunnlagt: <input type="text" name="grunnlagt"/>
  <input type="submit" name="lagre" value="Lagre"/>
</form>
```

endre\_by.php:

```
<?
if (isset($_REQUEST["lagre"])) {
  $by = $_REQUEST["by"];
  $grunnlagt = (int) $_REQUEST["grunnlagt"];
  // behandle data, lagre til database
  echo "Informasjon om " . $by . " er nå lagret.";
} else {
  // HTML output
  include("endre_by.html");
}
?>
```



## Cookies

Cookies brukes som regel til å gjenkjenne en bruker på klientsiden:

- en liten tekstfil med data som serveren sender til klienten
- klienten sender tilbake tekstfilen ved hver nye forespørsel til den gitte serveren

For å sette en cookie i PHP brukes:

```
setcookie(name, value, expire, path, domain)
```

```
setcookie("param1", "xxxxxxx", time() + 3600);
```

Variabelen `$_COOKIE` brukes for å hente ut dataene i en cookie:

```
$param1 = $_COOKIE["param1"];
```

Debug og fallgruver:

- cookies blir ikke tilgjengelig før neste lasting av siden
- `setcookie` må kalles før noe HTML-output har skjedd (fordi cookies settes i HTTP headeren)
- for å skrive ut alle cookies: `print_r($_COOKIE);`



# Alle globale array-variabler

Bruker / klient - input:

- `$_GET` - HTTP GET variabler
- `$_POST` - HTTP POST variabler
- `$_REQUEST` - begge over
- `$_COOKIE` - cookies
- `$_FILES` - opplastede filer fra HTML forms

Direkte tilgang til rådataene i inputen:

- `$HTTP_RAW_POST_DATA`

Serverkonfigurasjon:

- `$_SERVER` - web-server konfigurasjon
- `$_ENV` - environment variabler
- `$argv` - input-parametre hvis startet fra kommandolinje



# HTTP Header

Av og til ønsker man å gi direktiver direkte til browseren eller web-proxyer på veien fra serveren. Direktivene sendes i HTTP headeren før selve siden med HTML kommer og disse må da settes før det gjøres noe output i scriptet.

- Redirection

```
<? header("Location: http://example.com/"); ?>
```

- Last-Modified

```
<? header("Last-Modified: " .  
        date("d M Y H:i:s", getlastmod())); ?>
```

- Unngå all caching

```
<? header("Cache-Control: no-cache, must-revalidate"); ?>  
<? header("Pragma: no-cache"); ?>  
<? header("Expires: Mon, 26 Jul 1980 05:00:00 GMT"); ?>
```



# Sesjonshåndtering

HTTP er en tilstandsløs protokoll (med unntak av de små mengdene data man kan lagre i en cookie), men behovet for sesjonshåndtering i web-applikasjoner er absolutt til stede.

PHP løser dette ved hjelp av sesjonsvariabler lagret på serveren (i minnet som standard).

Sesjonen må settes opp ved hjelp av `session_start()` som lager en unik id (UID) for hver besøkende og et array (`$_SESSION`) som har plass for sesjonsvariablene. UID'en blir enten lagret i en cookie eller i modifiserte URL'er.



# Sesjonshåndtering

index.php:

```
<?
  session_start(); // Setter UID cookie og gir
                  // tilgang til $_SESSION

// er brukeren pålogget?
if (isset($_SESSION["userid"])) {
    include("main.php"); // bare for påloggede brukere
} else {
    include("login.php"); // for å vise pålogging
}
?>
```



# Sesjonsåndtering

login.php:

```
<?
session_start();
if (pålogging ok) {
    $_SESSION["userid"] = $userid;
}
?>
```

logoff.php:

```
<?
session_destroy();
?>
```



## Sesjonshåndtering

Det er mulig å spesifisere hvordan sesjonsdataene skal lagres på serveren:

```
session_module_name("files"); // temp filer  
session module name("mm"); // minne  
session_module_name("user"); // egne  
brukerfunksjoner
```

Egne brukerdefinerte funksjoner kan settes opp:

```
session_set_save_handler("myOpen", "myClose",  
"myRead", "myWrite", "myDestroy", "myGC");
```



# Filhåndtering

Det er mulig å lese fra lokale filer på serveren:

```
$filnavn = "filnavn.txt";  
$fil = fopen($filnavn, "r");  
$innhold = fread($fil, filesize($filnavn));  
fclose($fil);
```

Shortcut for å lese hele filen inn i en streng (PHP 4.3):

```
$innhold = file_get_contents($filnavn);
```

Skrive til filer:

```
$filnavn = "filnavn.txt";  
$fil = fopen($filnavn, "w");  
fwrite($fil, "data");  
fclose($fil);
```

Shortcut for å skrive en streng til en fil (PHP 5):

```
file_put_contents($filnavn, "data");
```



## Filhåndtering

Det er også mulig å lese filer fra andre steder ved bruk av innebygde "wrappers":

- HTTP og HTTPS
- FTP og FTPS
- input/output streams
- compression streams
- secure shells
- audio streams
- med flere: <http://no.php.net/manual/en/wrappers.php>

Man spesifiserer da bare ønsket protokoll i filnavnet:

```
$filnavn = "http://www.vg.no/";  
$dagens_vg = file_get_contents($filnavn);
```

```
$filnavn = "ftp://bruker:passord@dagbladet.no/index.html"  
file_put_contents($filnavn, $dagens_vg);
```



## Feilrapportering

Vanlig feilrapportering (E\_ALL) i PHP er ganske enkel:

- feilmelding med navn på kildekodefilen, linjenummer og en beskrivende feilmelding blir sendt til browseren:

```
Warning: fopen(filnavn.txt) [function.fopen]: failed to open stream:
```

```
No such file or directory in C:\workspace\coding\web\test.php on line 2
```

- scriptet fortsetter å kjøre
- graden av feil som skal rapporteres kan settes:

```
error_reporting(0); // slå av all rapportering
```

```
error_reporting(E_ERROR | E_WARNING); // slå på feil  
og advarsler
```

```
error_reporting(E_ALL); // slå på all rapportering
```



# Feilhåndtering

Alle returkoder fra funksjoner bør sjekkes og håndteres:

```
$fil = @fopen($filnavn, "r");  
if ($fil) {  
    // behandle fil  
} else {  
    // behandle feil  
}
```

@-tegn foran funksjonskall slår av feilrapportering lokalt for den funksjonen, selv om global feilrapportering er satt.



# Optimalisering av koden

- Ikke bruk reg-exp hvis du ikke trenger
  - BAD: `<? $new = ereg_replace("-", "_", $str); ?>`
  - GOOD: `<? $new = str_replace("-", "_", $str); ?>`
  - BAD: `<? preg_match('/(\\.\\.\\.*?)$/', $str, $reg); ?>`
  - GOOD: `<? substr($str, strrpos($str, '.')); ?>`
- Bruk referanser (pekere) hvis behov for å sende store datamengder mellom funksjoner
- Bruk persistente databasetilkoblinger hvis hastighet er viktig
  - en del databaser er tregere enn andre på å opprette nye connections, persistente tilkoblinger er oppe også når scriptet ikke kjører og er klar når request kommer inn
- For MySQL: bruk `mysql_unbuffered_query()`
  - returnerer med en gang uten å vente på at databasen har returnert alle data => raskere og bruker mindre minne
- Hvis du skriver mye kode for å gjøre en generell oppgave, så finnes det garantert en innebygd funksjon for det



# Debugging

PHP har ingen innebygd støtte for debugging, men det er mulig å bruke eksterne debuggere og utvidelser:

- Zend IDE
- DBG PHP Debugger and Profiler
- Advanced PHP Debugger (APD)
- Eclipse for PHP Developers

Som regel nøyer man seg med å sette `error_reporting(E_ALL)`, samt skrive ut variabler:

```
function debug_view($var) {  
    echo "<pre>";  
    if (is_array($var)) {  
        print_r($var);  
    } else {  
        var_dump($var);  
    }  
    echo "</pre>";  
}
```



# Eclipse for PHP Developers

Eclipse PHP Development Tools 2.1 (med støtte for PHP 5.3)

- PHP Project Wizard
- Source Editing
  - Syntax coloring and validation
  - Code Assist
- Code Navigation i PHP Explorer View
- Mark Occurrences
- Override Indicators
- Type Hierarchy
- Local and remote debugging

<http://eclipse.org/downloads/packages/>



# Installasjon av PHP sammen med Apache og MySQL

Hvis man vil prøve PHP på en webserver med database lokalt anbefales å installere en AMP pakke.

AMP = Apache Webserver, MySQL Database og PHP/Perl/Python

[http://en.wikipedia.org/wiki/List\\_of\\_AMP\\_packages](http://en.wikipedia.org/wiki/List_of_AMP_packages)

WAMP = AMP for Windows: <http://en.wikipedia.org/wiki/WAMP>

LAMP = AMP for Linux: [http://en.wikipedia.org/wiki/LAMP\\_\(software\\_bundle\)](http://en.wikipedia.org/wiki/LAMP_(software_bundle))

MAMP = AMP for Mac: <http://en.wikipedia.org/wiki/MAMP>

[http://en.wikipedia.org/wiki/Comparison\\_of\\_WAMPs](http://en.wikipedia.org/wiki/Comparison_of_WAMPs)

Velger EasyPHP fordi den oppdateres ofte (inneholder siste versjoner), er liten av størrelse (inneholder bare det mest nødvendige) og har enkel administrasjon.



# Installasjon av PHP sammen med Apache og MySQL

"EasyPHP automatically installs and sets up a work space: it is a development tool, not a production tool. If you want to work on production, the solution LAMP (Linux Apache MySQL PHP) will suit your needs."

- Installere og starte Apache og MySQL
- Konfigurasjon av Apache (httpd.conf)

```
LoadModule php5_module "${path}/php/php5apache2_2.dll"  
PHPIniDir "${path}/apache"  
AddType application/x-httpd-php .phtml .pwm1 .php5 .php4 .php3  
.php2 .php .inc
```
- Konfigurasjon av PHP (php.ini)

```
short_open_tag = Off  
max_execution_time = 30  
error_reporting = E_ALL | E_STRICT  
display_errors = On  
post_max_size = 8 MB
```



## Opprette en ny web-applikasjon

### Eclipse

- New PHP Project: kompis
- New PHP File: info.php
- info.php: `<? phpinfo(); ?>`

### EasyPHP

- Add Alias: kompis -> c:\coding\workspace\kompis
- php.ini: short\_open\_tag = On

### Browser

- <http://localhost/kompis/info.php>



## Demo: bruk av database

### Eclipse

- Open Project: demo

### EasyPHP

- Add Alias: demo -> c:\coding\workspace\demo

### phpMyAdmin / annet databaseverktøy

- Opprett ny database: demo
- Opprett tabell: sted { navn, grunnlagt, innbyggere }

### Browser

- <http://localhost/demo/>
  - db.php: generell oppkobling mot MySQL
  - install.php: setter inn eksempel-data for steder i Norge
  - view.php: viser stedene som er registrert
  - insert.php: sett inn nytt sted



## Demo: GD bibliotek

### Eclipse

- logo.php
  - bruker GD bibliotek for å lage et PNG-bilde
  - bruker font.ttf som skrifttype for å skrive ut klokkeslett
  - bruker systemek.jpg som logo

### Browser

- <http://localhost/demo/logo.php>



## Open Source PHP rammeverk

### Wordpress

- største rammeverk for blogging
- <http://wordpress.org/>

### Joomla

- mest populære CMS
- veldig utvidbart
- støtte for mange plugins og design
- <http://joomla.org/>

### Smarty

- enkel og rask "template engine"
- effektiv bruk av templates og tags
- <http://smarty.net/>



## Vel hjem

### Kilder:

- <http://no.php.net/>
- <http://no.php.net/manual/en/getting-started.php>
- <http://en.wikipedia.org/wiki/PHP>
- <http://w3schools.com/PHP/>
- <http://eclipse.org/downloads/packages/>
- <http://easyphp.org/>
- <http://ukuug.org/events/linux2002/papers/html/php/>