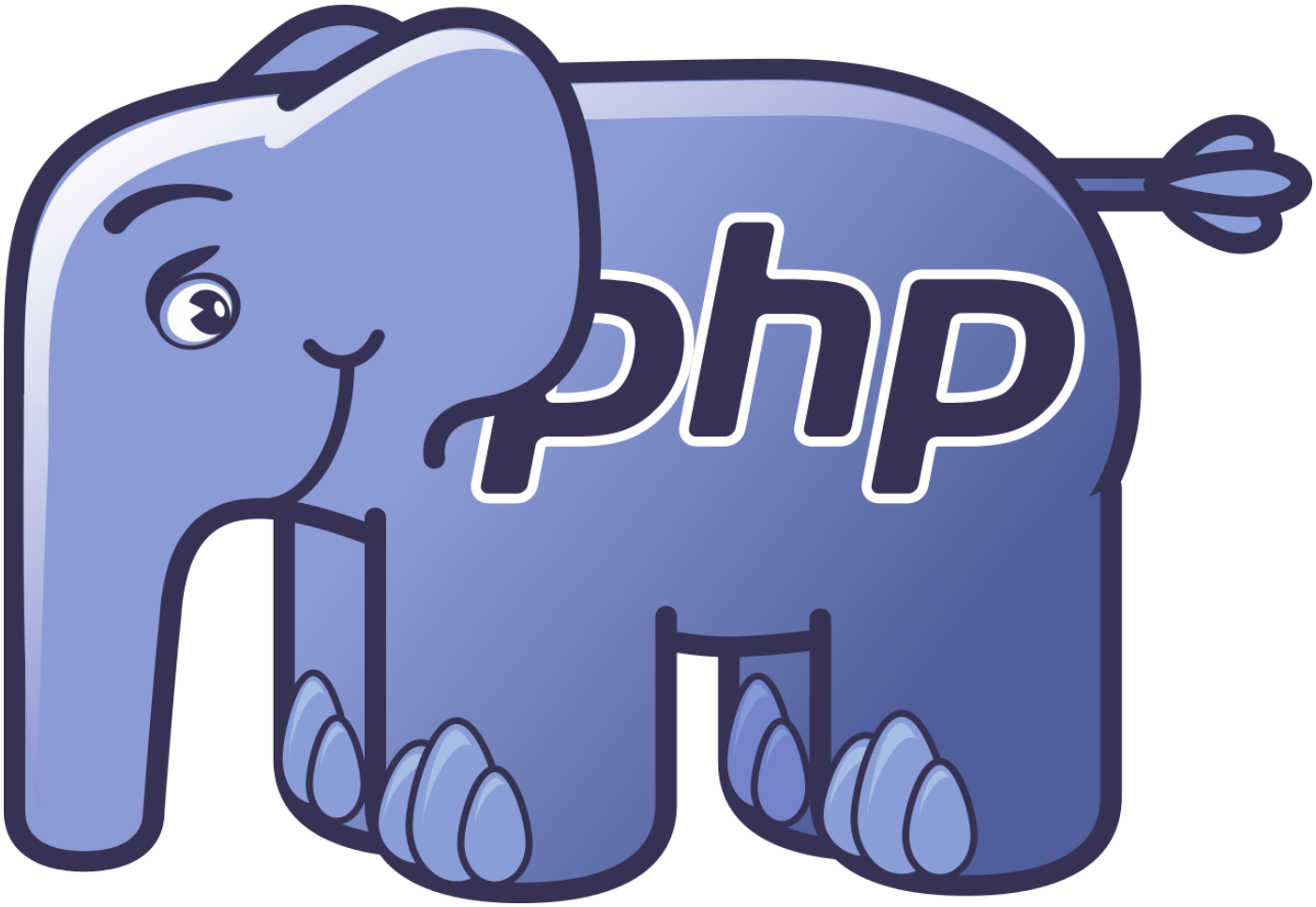
MODULE DYNAMIC WEB
CSG AUGUSTINUS



PHP & MySQL

INHOUDSOPGAVE

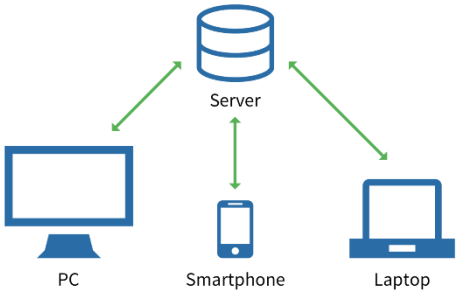
Inhoud

INLEIDING

In klas 4 heb je geleerd om websites te maken met behulp van **HTML** en **CSS**. Deze websites waren **statisch**. de bezoeker van de website kon niets aan de webpagina veranderen: er was geen **interactie**. Die interactie kwam er wel, toen je leerde programmeren in **Javascript**. Gebruikers konden aanpassingen doen met behulp van muis en toetsenbord: de webpagina was daarmee **interactief** geworden.

De overeenkomst tussen HTML, CSS en Javascript is dat deze alle drie door de browser worden verwerkt. Dat wil zeggen: op de computer van de gebruiker zelf. Er is dan wel enige interactie, maar die is alleen zichtbaar voor de gebruiker zelf. Veranderingen zijn tijdelijk: als de webpagina opnieuw wordt geladen of door iemand anders wordt geopend, is alles weer zoals het was.

Hiernaast zie je een schematische weergave van het **client-servermodel**. Gebruikers (*clients*) benaderen een centrale webserver via het internet. Deze webserver geeft HTML, CSS, Javascript (maar bijvoorbeeld ook afbeeldingen) terug aan de browser van de *client*. Aan de kant van de gebruiker, *client-side*, worden alle informatie verwerkt en als webpagina (of iets anders) getoond.



FIGUUR 1

Javascript is een cliënt-side scripttaal, omdat het aan de kant van de gebruiker wordt uitgevoerd. Er bestaan ook scripttalen die *server-side* worden uitgevoerd. **PHP** is daar een voorbeeld van. In deze module leer je programmeren in PHP en leer je interactieve websites te maken die blijvend veranderd kunnen worden door een gebruiker.

Deze veranderingen worden vaak opgeslagen in een database. Deze database bevindt zich ook op de webserver. Als database management systeem gaan wij **MySQL** gebruiken. In de vorige module heb je hier al kennis mee gemaakt.

Opdracht 1: client-server

1. Hoe zou jij het begrip *interactieve website* omschrijven?
2. Bedenk minimaal drie voordelen van een *server-side* scripttaal t.o.v. een *client-side* scripttaal.
3. Bedenk een voordeel van een *client-side* scripttaal t.o.v. een *server-side* scripttaal.

H1 BASIS PHP

1.1 PHP-bestanden en variabelen

Op de schoolwebsite staat voorbeeld_1.php. Als je de broncode van dit bestand bekijkt, begint het met:

```
<?php

// Commentaar kan gegeven worden met //

/*  Wie meerdere regels commentaar
    heeft kan gebruik maken van de /* (begin) met aan het eind */

error_reporting(E_ALL);                                  // Merk op dat elke regel eindigt met ;
                                                         // variabelen beginnen met een dollar-teken

$voornaam = "Augustinus";                                // tekst met dubbele aanhalingstekens
$achternaam = 'van Hippo';                               // tekst met enkele aanhalingstekens
$geboren = 354;                                          // integer
$leeftijd = 2017-$geboren;                               // variabele op basis van berekening
$dagen = $leeftijd*365;                                  // berekening
$volledige_naam = $voornaam." ".$achternaam;             // nieuwe string op basis van variabelen

?>
```

Het voorbeeld hierboven bevat vele kenmerkende elementen die we hier niet één voor één zullen bespreken. Je leert er wel mee werken in de opdrachten. Een paar hoofdpunten:

De server weet dat hij de code als PHP-code moet uitvoeren doordat deze tussen **<?php** en **?>** staat. Variabelen beginnen altijd met het dollar-teken **$** en elke regel eindigt met een puntkomma **;**. In veel gevallen bestaat op zijn minst een deel van de uitkomst van de programmeercode uit HTML-code die naar de cliënt wordt gestuurd. In voorbeeld 1 wisselen HTML en PHP (die leidt tot nieuwe HTML) elkaar af:

```
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Voorbeeld 1</title>
  </head>
  <body>
    <h1>
      <?php echo $volledigenaam; ?>
    </h1>
    <p>
      <?php
      // Binnen de html kun je op elk moment beginnen met een stukje PHP
      echo "Als hij nog zou leven, dan zou hij nu <i>grofweg</i> $dagen dagen oud zijn.";
      ?>
    </p>
  </body>
</html>
```

De HTML-uitvoer in dit voorbeeld zijn die regels die beginnen met **echo**. In voorbeeld 1 staan verschillende varianten van het werken met **echo** beschreven. Bestudeer ze nauwkeurig: kleine wijzigingen in de code kunnen grote gevolgen hebben!

Op www.php.net is een uitgebreide documentatie te vinden. Hier vind je ook uitleg bij het zeer grote aantal ingebouwde functies binnen PHP.

Opdracht 2: Teksten (strings) tonen en bewerken

4. Open *opdracht_2.php* in de editor en bekijk het resultaat in de browser.
5. Pas de regel *echo '<h4>$tekst</h4>';* aan, zodat de inhoud van de variabele *$tekst* (nogmaals) op het scherm wordt getoond. Doe dit zonder dat je dubbele aanhalingstekens gebruikt.
6. Haal de // weg voor regel 10 en bekijk het resultaat. Wat doet de functie *str_replace*?
7. Ga naar www.php.net en gebruik de zoekfunctie om uitleg te vinden bij de functie *strstr*. Verander de inhoud van de variabele *$tekst* naar (alleen nog) *De Albert Heijn betaalt goed* met de *strstr*.
8. Maak een variabele *$lengte* en gebruik de functie *strlen* om te bepalen uit hoeveel karakters de variabele *$tekst* nu bestaat. Zorg dat de volgende zin op het scherm wordt getoond, met op de plaats van het getal 13 het juiste aantal. Zorg voor schuingedrukte letters en een vetgedrukt aantal. *De string $tekst bestaat uit **13** karakters.*
9. Worden spaties meegeteld in de functie *strlen*?
10. Typ onder de vorige zin de PHP-regel *$tekst=substr($tekst,10,5);* en zorg dat de inhoud van de variabele *$tekst* daarna weer op het scherm wordt getoond.
11. Wat is de betekenis van de getallen 10 en 5?
12. Typ na de vorige PHP-regel *echo strrev($tekst).'
';* in en bekijk het resultaat.
13. Bij de zoekopdrachten hierboven op www.php.net verscheen er aan de rechterkant van het beeldscherm steeds een lijst met string-functies. Probeer een paar functies uit.

Opdracht 3: Waardes tonen en bewerken

14. Open *opdracht_3.php* in de editor en bekijk het resultaat in de browser. Dit is voorbeeld 2 van de schoolwebsite.
15. De waarde van *$afgerond* is bijna 1,2345678. We willen 1,234567 op het scherm krijgen, maar dat lukt niet door af te ronden met de functie *round*.
    Zoek via www.php.net op wat de functie *floor* doet en gebruik deze functie om (met behulp van *$rec*) de waarde 1,234567 op het scherm te krijgen.
16. Pas de regel echo '<h4>$tekst</h4>'; aan, zodat de inhoud van de variabele $tekst (nogmaals) op het scherm wordt getoond. Doe dit zonder dat je dubbele aanhalingstekens gebruikt.
17. De stelling van Pythagoras is te schrijven als $a^2 + b^2 = c^2$. Maak de variabelen *$a* en *$b* aan en geef deze zelf een waarde. Laat PHP vervolgens de waarde van *$c* berekenen en afronden op drie cijfers achter de komma. Toon het eindresultaat op het scherm. Zoek zelf via www.php.net naar een functie waarmee je de wortel kunt trekken.
18. Een bij programmeren veelgebruikte functie die je niet zo vaak bij wiskunde tegenkomt is de modulo-functie. Deze geeft de restwaarde bij een deling. Vorig jaar heb je hem misschien gebruikt bij het tonen van *sprites* in een game. De modulus is de restwaarde van een deling: 14 / 3 => 2. Zoek uit hoe de modulus in PHP kan worden berekend en voer een modulo-berekening uit.

Opdracht 4: Arrays

19. Bekijk voorbeeld 3 op de schoolwebsite.
20. Open *opdracht_4.php* in de editor en bekijk het resultaat in de browser.
21. Vul regel 9 aan, zodanig dat inhoud van het vierde element van de array wordt getoond.
22. In voorbeeld 3 worden twee technieken getoond om de volledige inhoud van een array te tonen. Probeer ze allebei uit voor de array *$priem*.
23. Zet de code van de vorige vraag tussen /* en */ zodat deze niet meer wordt uitgevoerd.
24. Gebruik de *count*-functie (zie eventueel www.php.net) om het aantal items in de array *$priem* te tellen. Gebruik daarna *echo* om alleen het laatste item van de array op het scherm te zetten.
25. Open *opdracht_4b.php* in de editor en bekijk het resultaat in de browser.
26. Pas de array *$leerling* aan, zodanig dat jouw gegevens worden opgeslagen, **inclusief** jouw geboortejaar.
27. Gebruik *echo* en de inhoud van de array om de zin *Alan Turing werd geboren in 1912*. op het scherm te tonen (maar dan met jouw gegevens).
28. Op www.php.net verscheen bij het zoeken naar *count* een lijst met array-functies. Probeer er twee.

1.2 Herhalingen en voorwaarden

In het lesmateriaal over Javascript heb je al kennis gemaakt met verschillende vormen van herhalingen en hun nut en toepassingen. Bij een vooraf bekend aantal herhalingen gebruiken we ook bij PHP een *for*-loop:

```
$aantal=5;
for ($teller=1;$teller<=$aantal;$teller++) {
  echo $teller." | ";
}
```

Deze syntax lijkt erg op die van Javascript. Dat geldt ook voor een herhaling met een *while*:

```
$controle=9;
while ($controle>=$aantal) {
  echo $controle." | ";
  $controle--;                          // verlaag de waarde van $controle met 1
}
```

Bij een *while* gebruik je een **voorwaarde**. Zolang er aan deze voorwaarde wordt voldaan, gaat de herhaling door. Het vetgedrukte deel van de code is in dit geval de voorwaarde. Na elke herhaling wordt gecontroleerd of de waarde van *$controle* nog steeds groter of gelijk is aan de waarde van *$aantal*. De uitslag van zo'n controle is **true** of **false**, net als bij een variabele van het type **boolean**.

In opdracht 4 heb je gewerkt met **arrays** of lijsten. Als je een bepaalde handeling voor elk element uit een array wilt uitvoeren, kun je ze één voor één bij langs lopen met *foreach*:

```
$getallenrij=array(2, 3, 5, 7, 11);
foreach ($getallenrij as $getal) {
  echo $getal." | ";
}
```

In bovenstaand voorbeeld krijgt de variabele *$getal* bij elke herhaling de waarde van een volgend element uit de array *$getallenrij* totdat het laatste element is bereikt. Je hoeft dus niet vooraf te tellen uit hoeveel items de lijst bestaat.

Opdracht 5: Oefenen met herhalingen

29. Open *opdracht_5.php* in de editor en bekijk het resultaat in de browser.
30. Pas de *for*-loop aan, zodanig dat de array *$reeks* gevuld wordt met de eerste acht **oneven** getallen.
31. Maak een nieuwe (lege) array aan met de naam *$macht3*.
32. Gebruik een *foreach*-loop om *$macht3* te vullen met waarden die de derde macht zijn van de getallen uit de array *$reeks* (dus: 1, 8, 27, etc.) en zorg dat *$macht3* op het scherm wordt getoond.
33. Gebruik een *while*-loop om de array *$macht3* te doorlopen, totdat een waarde wordt bereikt die groter of gelijk is aan 1000. Alle waarden die aan de voorwaarde voldoen moeten worden getoond.

Opdracht 6: Tweedimensionale arrays

34. Open *opdracht_6.php* in de editor en bekijk het resultaat in de browser. De code bevat een tweedimensionale array en een dubbele *foreach*-loop. Bestudeer beide totdat je ze snapt.
35. Pas regels 14 en 15 aan zodat er kloppende zinnen op het scherm verschijnen.
36. (EXTRA) De 9 getallen zijn er nu met de hand ingezet. Schrijf een herhaling waarmee een tweedimensionale array gevuld wordt met de getallen 1 t/m 25, zodanig dat hiermee een speelveld van 5 x 5 ontstaat. Pas de zinnen weer aan, zodanig dat ze kloppen.

1.3 Voorwaarden en keuzes

Een *while*-herhaling werkt met een voorwaarde: zolang hieraan is voldaan gaat de herhaling, potentieel oneindig, door. In de vorige paragraaf zag je een voorwaarde in de vorm van een vergelijking, maar je kunt hiervoor ook een **boolean** gebruiken:

```
$doorgaan=true;                 // $doorgaan is hier een boolean
while ($doorgaan) {
    // voeg code toe. Zorg dat op een gegeven moment geldt: $doorgaan=false om de loop te stoppen
 }
```

Vanuit Javascript ken je ook al voorwaarden met een *if* en een *else*. In PHP is de vorm bijna hetzelfde:

```
$doorgaan=true;
$getal=0;
while ($doorgaan) {
    $getal++;
    If ($getal>10) {
            $doorgaan=false;
    }
    else {
            echo “Het getal $getal is kleiner dan 10.”;
    }
 }
```

(Merk op dat we in dit geval ook in één keer *while($getal<=10)* hadden kunnen gebruiken!)

Let goed op de syntax: als twee dingen gelijk moeten zijn gebruik je == net als bij Javascript. Als het één OF het andere mag gelden, gebruik je ||.

Opdracht 7: Fibonacci-reeks

37. Open *opdracht_7.php* in de editor en bekijk het resultaat in de browser. Dit is voorbeeld 5 van de schoolwebsite.
38. Leg uit waarom regel 52 en 53 nodig zijn voor de goede werking van de code.
39. Vervang regel 8 door de regel *$fibonacci=array(0,1);*
40. De Fibonacci-reeks wordt gemaakt door twee voorgaande Fibonacci-getallen bij elkaar op te tellen. Gebruik een *for*-loop om de array *$fibonacci* verder te vullen met de eerste 30 Fibonacci-getallen.
41. Gebruik de *count*-functie om de lengte van *$fibonacci* te bepalen, om te checken of je inderdaad precies 30 getallen hebt gegenereerd.
42. Het eerste Fibonacci-getal dat deelbaar was door 2 en door 3 was 144. Pas de code aan, zodanig dat het eerstvolgende Fibonacci-getal dat deelbaar door 2 en 3 wordt gevonden. Je mag gebruik maken van het getal 144.

Extra opdracht 8: Kruisje – Rondje

43. Open *opdracht_8.php* in de editor en bekijk het resultaat in de browser. Dit is een variant op opdracht 6 met in plaats van getallen kruisjes en rondjes.
44. Er is een *for*-loop gemaakt die controleert of een speler een hele **rij** bezet heeft. Op dit moment is dat het geval en is **X** de winnaar. Pas de array aan zodat er niet horizontaal maar verticaal een winnaar is en bekijk het resultaat.
45. Breid de code uit, zodanig dat er ook wordt gecontroleerd of er verticaal een winnaar is.
46. Breid de code uit, zodanig dat er ook wordt gecontroleerd of er diagonaal een winnaar is.

1.4 Zelf PHP-functies maken

Inmiddels heb je kunnen zien dat er behoorlijk wat overeenkomsten zijn tussen Javascript en PHP. In het algemeen geldt dat veel programmeertalen dezelfde structuren hebben en je een tweede taal sneller leert. De overeenkomsten gelden ook voor het zelf maken van functies. We bespreken hier drie varianten:

```
function dobbelsteen1() {
  $worp=1+rand(0,5);                                    // random-functie: willekeurig tussen 0 en 5.
  echo "<h2>Dobbelsteen1 is op de $worp gevallen.</h2>";
}
dobbelsteen1();                                         // het aanroepen van de functie
```

Een functie maak je met **function**. Tussen de accolades komt de code die moet worden uitgevoerd als de functie wordt **aangeroepen**. Pas als dit gebeurt, wordt de code uitgevoerd.

Het resultaat van een functie kan ook worden opgeslagen in een variabele:

```
function dobbelsteen2() {
  $worp=1+rand(0,5);
  return $worp;
}
$uitkomst=dobbelsteen2();                               // het aanroepen van de functie
```

In bovenstaand voorbeeld wordt de waarde van de variabele *$worp* teruggegeven (*return*) en opgeslagen in de variabele uitkomst.

Het is mogelijk om aan een functie een waarde of **parameter** mee te geven. In onderstaand voorbeeld is *$aantal* de parameter en 3 het **argument**:

```
function gooi_N_worpen($aantal) {
  for ($n=0;$n<$aantal;$n++) {
    dobbelsteen1();
  }
}
gooi_N_worpen(3);                                       // het aanroepen van de functie met een parameter
```

Merk op dat de functie *gooi_N_worpen* weer gebruik maakt van de functie *dobbelsteen1*. Je kunt de ene functie dus aanroepen binnen de andere functie.

Opdracht 9: Dobbelstenenstatistiek

47. Open *opdracht_9.php* in de editor en bekijk het resultaat in de browser.
48. Leg uit wat er gebeurt in de regels 18 t/m 20.
49. Verander het aantal keren dat er met de dobbelsteen wordt gegooid naar 100?
50. (EXTRA) Schrijf een functie *bereken_hoogte* die als invoer de array *$frequentie* heeft en als resultaat een array teruggeeft met de hoogtes van het staafdiagram, zodanig dat de hoogste staaf 200 pixels hoog wordt. Pas hiervoor ook de regel *$hoogte*=.. verder aan.
    HINT: Met de functie *max* kun je de hoogst voorkomende waarde vinden.
51. (EXTRA) zet // voor de regel *echo $worp."* | *";* en verhoog het aantal worpen. Bij welk aantal worpen is de frequentie voor alle zes mogelijke uitkomsten bij benadering hetzelfde?
52. (EXTRA) In theorie is de kans voor alle zes de mogelijkheden hetzelfde, namelijk 1/6. Bedenk een stappenplan om de computer net zo lang met de dobbelsteen te laten gooien, totdat voor alle frequenties geldt dat de afwijking t.o.v. 1/6 deel minder dan 1% is. Je hoeft het programma niet te schrijven. (Loop je voor op de klas, dan is het een leuke uitdaging.)

H2 WERKEN MET DATA

2.1 Gegevens laden uit een MySQL database

In het vorige hoofdstuk hebben we een aantal keren een reeks met gegevens verwerkt die was opgeslagen in een array. De volgende stap is om gegevens uit een database te laden en weg te schrijven in PHP-variabelen, zodat we ze op een website kunnen tonen of er bewerkingen op kunnen uitvoeren.

Stap 1 in dit proces is het maken van een verbinding (connectie) tussen het PHP-bestand en de database. Hiervoor zijn meerdere technieken. In deze module kiezen we voor **mysqli** (procedureel):

```
$DBverbinding = mysqli_connect('localhost', 'MIJN_gebruikersnaam','gEhEiM', 'Weerstations');

$DBverbinding = mysqli_connect($servernaam, $gebruikersnaam, $wachtwoord, $database);
```

De databaseverbinding wordt opgeslagen in *$DBverbinding*. Om query's te kunnen uitvoeren, zijn vier gegevens nodig die je rechtstreeks (bovenste variant) of met variabelen (onder) kunt invoeren.

Stap 2 is het uitvoeren van een query:

```
$sql = "SELECT * FROM stations WHERE plaats='Groningen' ";                              // let op ' ' en " "
$records = mysqli_query($DBverbinding, $sql);

$type=array('luchtdruk','windkracht','temperatuur');
$sql = "SELECT * FROM meters WHERE NOT merk='$merk' AND type='{$type['0']}' ";   // query met variabelen
```

De query is hier eerst in de variabele *$sql* gezet, maar kan ook rechtstreeks in de functie **mysqli_query** worden gezet. Wil je binnen de query gebruik maken van een string of array, dan moet je goed oppassen met aanhalingstekens. Voor arrays kan het handig zijn om accolades te gebruiken.

Stap 3 is om de in *$records* opgeslagen lijst met resultaten te vertalen naar een array per rij:

```
if (mysqli_num_rows($records) > 0) {
        while($record = mysqli_fetch_assoc($records)) {
          echo "Beheerder ".$record["beheerder"]." werkt in ".$record["plaats"].".<br>";
        }
}
```

Met de **while** en **mysqli_fetch_assoc** ga je één voor één langs de resultaten. In dit voorbeeld wordt elke rij uit de resultaattabel opgeslagen in *$record* (zonder –s). Om de inhoud van een resultaat te tonen, gebruik je de kolomnaam uit de tabel van de database, zoals *$record["beheerder"]*.
Opmerking: de **if**-constructie controleert of er wel resultaten zijn. Deze toevoeging kan eventueel achterwege gelaten worden.

Stap 4: als laatste sluit je de gemaakte verbinding met **mysqli_close**:

```
mysqli_close($DBverbinding);
```

Opmerking: het doel van deze beknopte uitleg is niet om diepgaand inzicht in alle genoemde functies te verkrijgen. Het doel van deze paragraaf is dat je op basis van de voorbeelden praktisch aan de slag kan met PHP en MySQL.

Opdracht 10: Gegevens laden uit een database

53. Open *opdracht_10.php* in de editor en bekijk het resultaat in de browser. Merk op dat de variabelen die nodig zijn om een databaseconnectie te maken uit een apart bestand worden gehaald (regel 9).
54. Pas de code aan, zodanig dat alle liedjes van de band *Nirvana* worden getoond die in 2014 in de lijst stonden met hun titel en de positie in 2014. Zorg dat de titels vetgedrukt worden geschreven.
55. Open *opdracht_10b.php* in de editor en bekijk het resultaat in de browser. Dit is resultaat van de vorige vraag, maar dan zonder allerlei toevoegingen.
56. In welke straat woon jij? Maak een variabele *$straat* en geef jouw straatnaam als waarde mee.
57. Gebruik de database *Postcode* om een overzicht te maken van alle plaatsen in Nederland waar een straat is met dezelfde straatnaam als waar jij woont. Gebruik in je query de variabele *$straat*. Zorg dat elke plaatsnaam maar één keer voorkomt en dat de namen alfabetisch gesorteerd zijn.
58. Zorg dat in de *while*-loop een array *$plaatsnamen* wordt gevuld met de gevonden plaatsen.
59. Gebruik een *foreach*-loop om alle elementen van *$plaatsnamen* op het scherm te tonen.
60. (EXTRA) De schrijver van deze module groeide op in de plaats *Klijndijk*. Dit dorp heeft maar een beperkt aantal straten. Genereer een overzicht van deze straatnamen met daarachter het aantal plaatsen in Nederland waar diezelfde straatnaam voorkomt. Gebruik hiervoor het volgende stappenplan:
    - I Maak een lege array *$straatnamen* aan.
    - II Schrijf een query die als resultaat alle unieke straatnamen in *Klijndijk* geeft.
    - III Gebruik een *while*-loop om de array *$straatnamen* te vullen met deze straatnamen.
    - IV Maak een *foreach*-loop die door één voor één de elementen van *$straatnamen* doorloopt
    - V Voer voor elke loop een query uit die het aantal plaatsen met die straat telt.
    - VI Toon voor elke loop de straatnaam met dit aantal op het scherm (met echo)

Opdracht 11: Aanpassingen doen aan een database

61. Open *opdracht_11.php* in de editor en bekijk het resultaat in de browser.
62. Bestudeer voorbeeld 9 van de schoolwebsite. Dit voorbeeld bevat query's die bewerkingen uitvoeren op een bestaande tabel.
63. Voeg jezelf toe aan de database *weerstations* als beheerder in je eigen woonplaats met behulp van een query in PHP.
64. Gebruik de functie *toon_tabel* om vast te stellen dat je bent toegevoegd aan de database.
65. Verwijder (via PHP) alle beheerders uit *Groningen* en controleer je werk met *toon_tabel*.
66. Experimenteer met het toevoegen, aanpassen en verwijderen van records in de andere twee tabellen van de database.
67. Om de database *weerstations* weer in zijn oorspronkelijke staat herstellen, open je *leesmij.html* en klik je op *Databases opnieuw aanmaken*. Kies de tweede actie uit de lijst.

2.2 Inloggen en sessies

Bij hoeveel websites heb jij een account? Dit zijn sites waar de inhoud van de webpagina – dus *wat je ziet* – afhangt van *wie* er is ingelogd (en *of* je bent ingelogd). Hoe gaat het verkrijgen van toegang in zijn werk?

Om in te kunnen loggen heb je een plek nodig waar je jouw naam en wachtwoord kunt intypen. Dit heet een **formulier**. Dit is de HTML-code van het formulier uit *voorbeeld 10*:

```
<form method="POST" action="">
        <label>Gebruiker</label>
        <input type="text" name="gebruikersnaam" placeholder="voer uw gebruikersnaam in...">
        <label>Wachtwoord</label>
        <input type="password" name="wachtwoord" placeholder="Geef uw wachtwoord...">
        <input type="submit" name="submit">
</form>
```

We zien dat een formulier gemaakt wordt met het <form>-element. Aan een formulier kun je vervolgens soorten (*type*) van invoer toevoegen. Hierboven zijn dit alleen een invoerveld (*type="text"),* een geheime invoer (type="password") en een verzendknop (type="submit"), maar er zijn ook andere vormen nodig zoals een *dropdown*-menu of een afvinklijst.

Alle invoer-elementen krijgen een naam (*name*) mee, waaronder de ingevoerde informatie wordt verstuurd. We sturen onze informatie met de methode POST, die we hier verder niet zullen toelichten. Aan het formulier kun je nog een *action* meegeven. Tussen de “” zou je bijvoorbeeld *ingelogd.php* kunnen schrijven. Na het verzenden van het formulier, gaat de site dan naar de nieuwe pagina *ingelogd.php*. Vul je niets in, dan worden de gegevens verzonden en wordt de pagina waar je al was opnieuw geladen.

Als de pagina (opnieuw of voor de eerste keer) wordt geladen, moeten een aantal zaken worden gecheckt:

- Is er iets verzonden met het formulier?
- Als er iets verzonden is met het formulier, moet gekeken of de inloggegevens correct zijn.
- Als er nog niets verzonden is of als de gegevens fout zijn, dan moet het formulier worden getoond.
- Zijn de inloggegevens wel correct, dan moet de gebruiker worden doorverwezen naar het afgeschermde deel (of er moet een andere weergave van dezelfde pagina worden getoond).

Deze vragen kun je stellen met **if** en **else**, maar wàt moet je nu precies vragen? In *voorbeeld 10* staat:

```
If (isset($_POST['gebruikersnaam'])) {
   $naam=$_POST['gebruikersnaam'];
   $pass=$_POST['wachtwoord'];
   $sql="SELECT * FROM stations WHERE beheerder='".$naam."'AND plaats='".$pass."' LIMIT 1";
   $records = mysqli_query($DBverbinding, $sql);
   if (mysqli_num_rows($records) == 1) {
     $_SESSION["user"]= "$naam";
     header("Location: voorbeeld_10.php");
     exit();
   }
   else {
     echo "<h1 style='color: red;'>Gebruikersnaam of wachtwoord onjuist</h1>";
   }
}

if (isset($_SESSION["user"])) {
  echo "<h1 style='color: green;'>Welkom ".$_SESSION["user"]."</h1>";
}
else {
  toon_formulier();
}
```

Op de volgende pagina bekijken we stap voor stap de belangrijkste code-elementen.

Op het moment dat op de verzendknop van het formulier wordt geklikt, wordt automatisch een array **$_POST** gevuld met de verzonden gegevens. In ons geval zijn dit *($_POST['gebruikersnaam']))* en *v($_POST['wachtwoord']))*. De functie **isset** kan controleren of een variabele een waarde heeft (dus niet *null*). De vraag *Is er iets verzonden met het formulier?* kunnen we dus beantwoorden met:
`If (isset($_POST['gebruikersnaam']))`

Als het antwoord op de vraag *ja* is, wordt een query uitgevoerd met code die je al kent. Op het moment dat precies één record wordt gevonden, is er een correcte combinatie van gegevens ingevoerd. Er volgt dan:

```
$_SESSION["user"]= "$naam";
header("Location: voorbeeld_10.php");
exit();
```

Wat is een **sessie** (session)? Als jij ingelogd bent op een site, moet het systeem onthouden dat jij bent ingelogd, OOK als je naar andere webpagina’s van die site gaat (en er dus een nieuwe pagina wordt geladen). Al jouw handelingen horen dan bij een sessie. Jouw toegangsrechten moeten in stand gehouden worden, totdat je uitlogt (of als je sessie is verlopen; op veel sites is dit het geval als je te lang *niks doet* op

de site). Hiervoor is de sessie-variabele **$_SESSION**. In de code wordt deze array gevuld met een element *user*. Omdat het inloggen gelukt is, moet nu de pagina opnieuw worden geladen (met **header**). Het uitvoeren van de code op de huidige pagina moet daarom worden onderbroken (**exit**).

Als je *voorbeeld_10.php* bekijkt zie je bovenaan het PHP-deel:
```
session_start();
```

In dit voorbeeld is er maar één pagina. In het algemeen geldt dat je deze regel moet toevoegen aan het begin van elke pagina die binnen de sessie bereikbaar moet zijn. Natuurlijk voeg je dan ook een controle toe of iemand wel ingelogd is!

Opdracht 12: Inloggen en uitloggen

68. Open *opdracht_12.php* in de editor en bekijk het resultaat in de browser. Dit is (ongeveer) de code van *voorbeeld 10*.
69. De tabel met accounts wordt op dit moment ook getoond als je al ingelogd bent. Dat is niet nodig. Zorg dat de tabel alleen wordt getoond, als er niemand is ingelogd.

Iemand die ingelogd is, wil ook weer kunnen uitloggen. Anders gezegd: hij wil de sessie kunnen beëindigen. Dit kun je met een verzendknop (type="submit") doen binnen een formulier, net als bij het inloggen.

70. Open *opdracht_12**b**.php* in de editor en bekijk het resultaat in de browser. Dit is een implementatie van een *logout met formulier*. Controleer de juiste werking.
71. Waar en onder welke voorwaarden wordt de functie *toon_logout* aangeroepen?

De getoonde is een beetje omslachtig. Meestal wordt in plaats daarvan een link getoond (al dan niet gekoppeld aan een plaatje of button) die verwijst naar een apart bestand *logout.php*.

72. Open *logout.php* in je editor.
73. Pas de functie *toon_logout* aan zodat er een hyperlink wordt getoond die verwijst naar *logout.php*.
74. Controleer de werking. Welke extra aanpassingen moet nog worden gedaan om het uitloggen op deze manier te laten functioneren?

Opdracht 13: Sessies over meerdere pagina's

75. Open *opdracht_13.php* en bekijk het resultaat in de browser.

Als je inlogt zie je dat je daarna binnen de sessie kunt kiezen voor meerdere pagina's. Deze zijn allen opgebouwd in een apart PHP-bestand.

76. Zorg dat je uitlogt, zodat de sessie beëindigd is. Klik daarna in de browser op de knop die ervoor zorgt dat je terug gaat naar de vorige pagina. Wat zie je?
77. Open *opdracht_13A.php* in de editor en zoek uit hoe het resultaat van de vorige vraag is bereikt.

Tot slot: veilig inloggen

De voorbeelden die je hier hebt gezien zijn alleen bedoeld ter inleiding op de vraag hoe je in kunt loggen en gebruik kunt maken van sessies. Afhankelijk van het project dat je uiteindelijk kiest kan het verstandig zijn om na te denken aan punten als:

- Het zelf laten aanmaken van accounts door de bezoekers van jouw website
- Het beveiligen van de login. Zie b.v. https://www.w3schools.com/php/php_form_validation.asp of http://thisinterestsme.com/secure-php-logout
- Het gebruik van encryptie binnen jouw database zodat jij als beheerder ook niet de wachtwoorden van jouw gebruikers kan lezen en niet alle wachtwoorden *op straat liggen* als je toch wordt gehackt.