Server Side Scripting

with PHP 5 and MySQL 5

Version 2013/2014

Sources

- Einstieg in PHP 5.5 und MySQL 5.6, *Thomas Theis, ISBN 978-3-8362-2489-5*

Different chapters of this document contain references to this book.

- W3Scools, *http://www.w3schools.com*
- PHP.net, *http://www.php.net*

Author

- Robert Fisch

Thanks to all students of the class T2IF1/2013-2014 for detecting and telling me about all errors and mistakes that sneaked off to this document.

The present document is under permanent development depending on your contributions. Ideas, hints and suggestions can be submitted to Robert Fisch who will take care to integrate them as soon as possible.

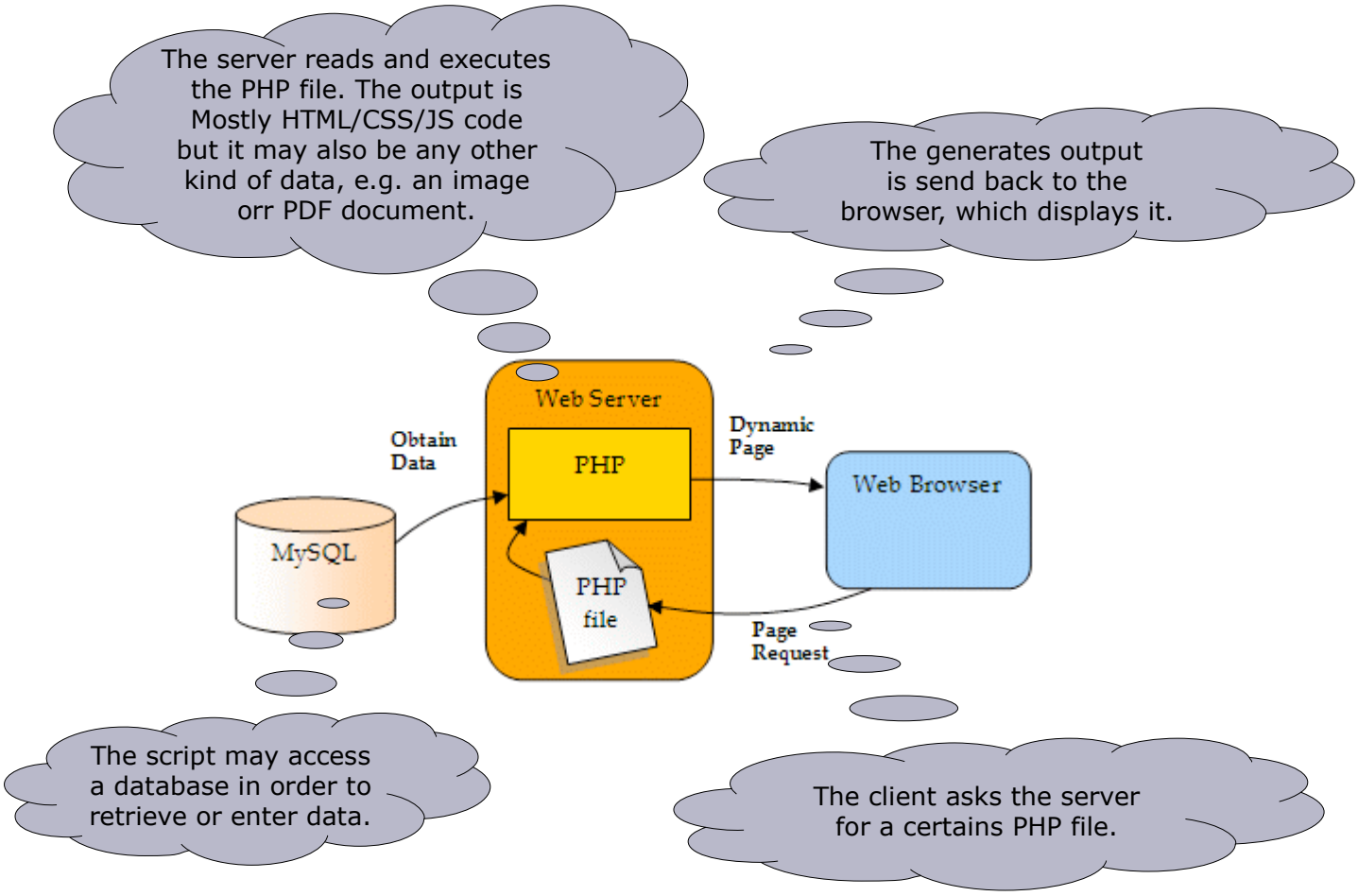
Table of contents

1. Getting started

1.1. Environment

In contrast to the developpment of client side scripting with Javascript, which runs inside the browser, the developpment of server side scripting needs the installation of a server. This will typically be an Apache + MySQL + PHP environment. These three pieces of software are often referred to as "AMP" environment.

- Apache is the web server
- MySQL is the database server
- PHP is the hypertext preprocessor



1.2. Linux installation

The here described installation is valid for any Debian or Debian based distribution. Use the folling commands to install Apache, MySQL and PHP:

```
apt-get update
apt-get install php5 php5-mysql apache2 mysql-server
```

During the installation process, you will have to set the root password for the MySQL database. After the installation, edit the file **/etc/php5/apache2/php.ini** to reflect the following changes:

```
short_open_tag = off
error_reporting = E_ALL
```

For PHP versions older than 5.4, also make shure the following settings are applied:

```
register_globals = off
safe_mode = off
```

The, reastart the Apache server:

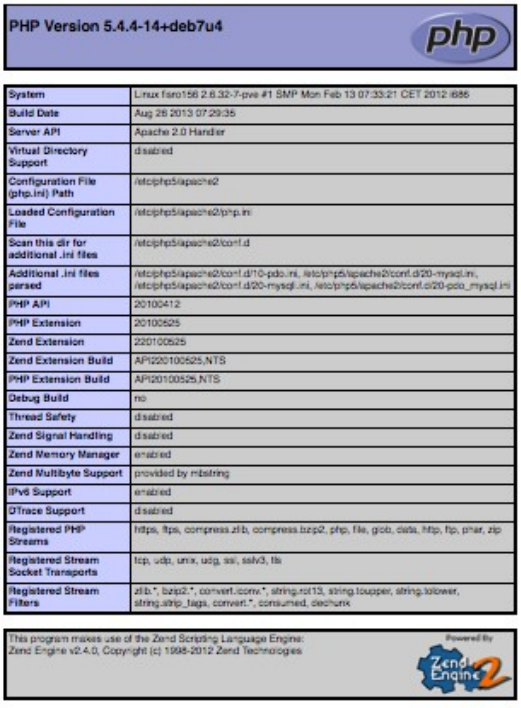
```
apache2ctl restart
```

To test if the web server is up and running, point your browser to the IP or name of your server. You should see something like "If works!".

To test if PHP is fine, do the following:

```
rm /var/www/index.html
echo "<?php phpinfo(); ?>" > /var/www/index.php
```

This creates the file **/var/www/index.php** and put a special command inside it, whichh gives all informations about the installed PHP. If you point your browser again at the IP or name of your server, you will see something like this:



1.3. Windows installation

Go and get any of the above packages, download the installer an run it on your Windows box. Just follow the on-screen instructions!

- MAMP http://www.mamp.info
- XAMMP http://www.apachefriends.org/en/xampp.html
- WAMPSERVER http://www.wampserver.com/en/

2. The basics

2.1. *Hello world!*

Of course, you need to write the classical "Hello world" script.

```
<!DOCTYPE html>
<html lang="en">
     <head>
          <meta charset="utf-8">
          <title>Hello world!</title>
     </head>
     <body>
          <?php
               echo "Hello world!";
          ?>
     </body>
</html>
```

PHP Code is embded into HTML files. "<?php" is the opening tag. "?>" is the closing tag. You have to write your PHP code between these tags.

"Hello world!" is a string.

"echo" is a construct that allows us to output things ...

After being executed, the following code is besing send back to the browser:

```
<!DOCTYPE html>
<html lang="en">
     <head>
          <meta charset="utf-8">
          <title>Hello world!</title>
     </head>
     <body>
          Hello world!
     </body>
</html>
```

Notice

- The browser does not "see" the PHP code, as it is being replaced by the corresponding output directly on the server, just before sending the file to the client.

Practice

1. Write a page that displays your name.
2. Write a page that outputs the result of the following expression: 2+3*4
3. Write a page that echos the following code the the output:

   <img id="fish" src="http://www.fisch.lu/f.gif" alt="a fish">

   What does the browser see? Take a look at the generated HTML code!

2.2. Variables

As for Javascript, you can define and use variables in PHP. Although it is quite the same, there are still some differences:

Example

```
<!DOCTYPE html>
<html lang="en">
     <head>
          <meta charset="utf-8">
          <title>Variables</title>
     </head>
     <body>
          <?php
               $name = "Bob";
               echo "Hello " . $name;
          ?>
     </body>
</html>
```

In PHP, variables ***always*** start with a dollar symbol. There is no need to prepend a "var"...

To concatenate two strings, you must use the "." operator!

String are either enclosed in double quotes or in single quotes, just like in Javascript.

Practice

1. Try the above example!
2. Make a copy of the previous exercice and replace the "." by a "+" and take a closer look at the output.
3. Write a page that calculates the sum of three variables. First, put integer values in the variables. Next, put real values in them and take a close look at the output.
4. Write a page that ouputs the content of the variable **`$test`** without having assigned it a value. *What happens?*

Regarding the last practice, you should learn to use the following functions:

- **`isset($someVariable)`**

  Tests if the variable **`$someVariable`** is set (assigned) or not.

  Example

  ```
  if (isset($name))  echo 'The variable $name has the value "'.$name.'"';
  else               echo 'The vairable $name has not been set!';
  ```

- **`unset($someVariable)`**

  Use this function if you want to unset a variable, so if you want it to be destroyed.

2.3. IF statement

Using an "if" is as easy as in Javascript:

Example

```
<!DOCTYPE html>
<html lang="en">
     <head>
          <meta charset="utf-8">
          <title>If</title>
     </head>
     <body>
          <?php
               $a = 2;
               $b = 3;
               if ($a>$b)
               {
                    echo "$a is bigger than $b";
               }
               else
               {
                    echo "$a is smaller than $b";
               }
          ?>
     </body>
</html>
```

The comparison operators are the same as in Javascript.

In order to compare thinks, we can use one of the following operators:

PHP		JS
<	less than	<
<=	less or equal than	<=
>	more than	>
>=	more or equal than	>=
==	equal than (for non-boolean values)	===
!=	not equal than (for non-boolean values)	!==
===	equal than (for boolean values)	===
!==	not equal than (for boolean values)	!==

To combine multilple condition into one if statement, you can use the logical operators "**&&**" (and) and "||" (or).

Practice

1. Try the above example! *Why does the output not contain **$a** but is content?*
2. Make a copy of the above example and replace the double quote by single quotes, the take a look at the ouput. *What has changed?*

2.4. Ternary operators

Sometimes, we don't want to write a lot of lines just to output something very simple. Let's consoder the following PHP code:

```
$lines = 3;
if ($lines==1) echo "You have $lines line";
else           echo "You have $lines lines";
```

Notice the additional "s"!

This could be brought down to the following code:

```
$lines = 3;
echo "You have $lines line".($lines!=1?"s":"");
```

This is called "inline if", "shorthand if" or "ternary operators". It works pretty the same way as a standard "if" statement, but is a lot shorter. Please notice, that the "else" part, so the ":" **is** mandatory!

2.5. The FOR loop

The "for" loop works exactely the same way as in Javascript. Take a look at the above example, which determines if a number is prime or not.

Example

```
// define the number to analyse
$number = 27;
// initialise the counter
$count = 0;
// loop
for ($i=1 ; $i<=$number ; $i++)
{
     // test if the actual $i is a divisor of $number
     if ($number % $i ==0)
     {
          // increment the counter
          $count++;
     }
}
// display the result
if ($count==2) echo "$number is prime";
else           echo "$number is not prime";
```

The variable `$i` is being defined.
Its initial value is 1.
The loop continues as long as its value is less or equal the value of `$number`.
`$i` is incremented each step by one.

Practice

1. Write a page that outputs 1000 time the message "Hello world!". Each "Hello world!" has to be written in a new line!
2. Copy the previous exercice and modify it, so that each line is prepended with it's line number, e.g. "27) Hello world!".
3. Copy the previous exercice and modify it, so that each line which line number is a multiple of 13 is being displayed in red.
4. Write a page with a PHP script that calculates the sum from A to B. A and B are two integer numbers which have to be hardcoded.

2.6. The WHILE loop

The "while" loop works exactely the same way as in Javascript. Take a look at the above example, which determines if a number is prime or not.

Example

```
// define the number to analyse
$number = 27;
// initialise the counter
$count = 0;
// initialise the loop variable
$i=1;
// loop
while($i<=$number)
{
     // test if the actual $i is a divisor of $number
     if ($number % $i ==0)
     {
          // increment the counter
          $count++;
     }
     // increment the loop variable
     $i++;
}
// display the result
if ($count==2) echo "$number is prime";
else           echo "$number is not prime";
```

Practice

1. Write a script that applies the following calculation to a hardcoded given number until the result is 1:
   - if the number is even, divide it by 2
   - else, meaning if the number is odd, multiply it by 3 and add 1.

   Your script has to count the steps needed for the initial number to reach 1. The number is hardcoded at the top of the script.

   Example: (10) → 5 → 16 → 8 → 4 → 2 → 1 = 6 steps

2.7. Lists

As in Javascript, there exists the notion of "lists", or "array" in PHP. They are used really widely, so it's very important to know about them. Contrary to Javacript, arrays in PHP are key bound, meaning that it's actually a map of {key,value} pairs. The key as well as the value might be of any type!

Example:

```php
$myCars=array();                    // this defines an empty array/list
$myCars[0]="Saab";
$myCars[1]="Volvo";
$myCars[2]="BMW";

$myCars=array(
     0 => "Saab",
     1 => "Volvo",
     2 => "BMW"
);

$myCars=array("Saab","Volvo","BMW");

echo "<pre>".print_r($myCars,true)."</pre>";
```

Use this line to display the an array nicely formated on the screen.

The key may also be a text:

```php
$students = array(
     "19900423177" =>   "Jos Schmit",
     "19861217166" =>   "Mariette Kohl"
);
```

Lists can contain other lists:

```php
$students = array(
     "19900423177" =>   array(
                              "firstname"       => "Jos",
                              "name"            => "Schmit"
                              ),
     "19861217166" =>   array(
                              "firstname"       => "Mariette",
                              "name"            => "Kohl"
                              )
);
```

Practice

1. Define in PHP a phonebook. A phonebook is a list of persons in which each person has a name, a phone number and an emailaddress. Your phonebook must contain at least 3 entries.

2.8. Looping through lists

For integer index lists, where you are shure that the keys are in order, you may use a usual FOR loop:

```
$myCars=array("Saab","Volvo","BMW");
for($i=0;$i<count($myCars);$i++)
{
     echo "Car N° $i = ".$myCars[$i]."<br>";
}
```

If there is a gap in the list, this kind of loop does ***not*** work anymore!

```
$myCars=array(
     0 => "Saab",
     2 => "Volvo",
     3 => "BMW"
);

for($i=0;$i<count($myCars);$i++)
{
     echo "Car N° $i = ".$myCars[$i]."<br>";
}
```

The function `count()` is used to determine the number of elements in an array.

In this cas we need to use the FOREACH loop:

```
$myCars=array(
     0 => "Saab",
     2 => "Volvo",
     3 => "BMW"
);

foreach($myCars as $key=>$value)
{
     echo "Car N° $key = ".$value."<br>";
}
```

If you don't need it, you may ommit the **`$key`** variable.

```
$myCars=array(
     0 => "Saab",
     2 => "Volvo",
     3 => "BMW"
);

foreach($myCars as $value)
{
     echo "Car = ".$value."<br>";
}
```

The names **`$key`** and **`$value`** a free to chose, but the here proposed names mostly fit well.

3. Inputs

Referring to the IPO[1] method, all our scripts actually had stripped of the input because we hardcoded all values. During the next chapters, you will learn how to get user data inside your PHP scripts.

3.1. The GET method

A first and easy, but not quite secure way, to pass input data to a PHP web script, is to call it using HTTP parameters. In the webbrowser, this might look like this:

```
http://www.ltam.lu/index.php?menu=269&page=103&portal=26
```

- This URL contains three parameters: menu, page & portal
- The respective values are: 269, 103 & 26
- There must be a "?" between the name of the PHP document and the first parameter.
- Parameters are separated unsing the "&" sign.

Still referring to the URL given above, we can retrieve the GET-parameters in PHP like this:

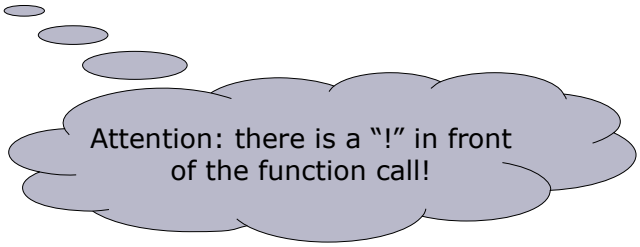
```
echo "The parameter 'menu' has the value: ".$_GET['menu'];
echo "The parameter 'page' has the value: ".$_GET['page'];
echo "The parameter 'portal' has the value: ".$_GET['portal'];
```

As you probabely noticed, there exists a special variable – which is an "array" - that holds all the passed parameters. We could then use a FOREACH loop (e.g. to debug a scripts input):

```
foreach($_GET as $key=>$value)
    echo "The parameter '$key' has the value: ".$_GET[$value];
```

If your script relies on a special parameter, you can use the **`isset()`** method to check if a given parameter has been set of not:

```
if(!isset($_GET['portal'])) $_GET['portal']=1;
```



1 http://en.wikipedia.org/wiki/IPO_Model

Practice

1. Write a page that outputs the message "Welcome!" as many times as the GET parameter **`loops`** indicates.
   - If **`loops`** has no value, it must have the default value of 13.
   - Each "Welcome!" has to be written in a new line.
   - The lines must be numbered.
   - Lines which number is a multiple of 13 must be dispayed in yellow. Those who are a multiple of 7 must be colored in blue. If a line number is a multiple of 7 and 13, draw it in green!
2. Write a page with a PHP script that calculates the product from A to B. A and B are two integer numbers which are given as parameters to the script.
3. Write a page, that outputs all parameters which have been passed to the script in form of a key-value table. The format of this table has to be the one given here:

Key	Value

   The length of the table depends of course on the number of parameters.
4. Write a script that applies the following calculation to the GET parameter **`number`** until the result is 1:
   - if the number is even, divide it by 2
   - else, meaning if the number is odd, multiply it by 3 and add 1.

   Your script has to count the steps needed for the initial number to reach 1.

   Example: (10) → 5 → 16 → 8 → 4 → 2 → 1 = 6 steps

3.2. *Modularizing pages using "include"*

Most of the time, developers are lazy and don't want to rewrite the same code multiple times. Next, if there is something to change, they prefer to change it only once. This is why copy & pasting code around isn't a good solution. The best way in PHP to reuse code is by using the **include** or **include_once** functions.

3.2.1. The « include » function

The **include** statement includes and evaluates the specified file. The generate code is put at the place where the call is being made.

Example

The output of the file "header.php" will be put here.

```
<!DOCTYPE html>
<html lang="en">
     <head>
          <meta charset="utf-8">
          <title>If</title>
     </head>
     <body>
          <?php
               include('header.php');
          ?>
          <p>A paragraph ... </p>
     </body>
</html>
```

If the file "header.php" has this content:

```
<?php
     echo "Welcome stranger!"
?>
```

The output that is being seng to the browser is as follows:

```
<!DOCTYPE html>
<html lang="en">
     <head>
          <meta charset="utf-8">
          <title>If</title>
     </head>
     <body>
          Welcome stranger!
          <p>A paragraph ... </p>
     </body>
</html>
```

3.2.2. The « include_once » function

The **include_once** statement includes and evaluates the specified file during the execution of the script. This is a behavior similar to the include statement, with the only difference being that if the code from a file has already been included, it will not be included again. As the name suggests, it will be included just once.

include_once may be used in cases where the same file might be included and evaluated more than once during a particular execution of a script, so in this case it may help avoid problems such as function redefinitions, variable value reassignments, etc.

Practice (for chapter 3.2)

1. Write a small homepage which respects the following constraints:
   - The subdirectory "Inlcude" contains 4 files: a.php, b.php, c.php and d.php which output simply the letters "a", "b", "c" and "d" respectively.
   - The main file (1.php) contains 4 links to itself. Each of these links has a different parameter. The parameter must have the name "include" and the values "a", "b", "c" and "d" respectively.
   - Depending on the value of the parameter "include", the main file has to include the respectively file from the "Include" folder.

Hints

- Use **$_SERVER['SCRIPT_NAME']** to retrieve the name of the actual script! This makes your homepage more portable …
- For making your script more secure, you may want to use the function **file_exists($filename)**, which tests if a given ile exists or not.
- For even more security, you may use the funciton **realpath($filename)** and then check that the destination file is contained in the right directory.

3.3. The POST method

As described in the book, the input via the POST method requires the use of an HTML form. There exists different type of inputs:

- textfield `<input type="text" name="DATA_a_textfield">`
- radio button `<input type="radio" name="DATA_a_button">`
- checkbox `<input type="checkbox" name="DATA_a_button[]">`
- list

```
<select name="DATA_a_list">
    <item value="a">A</item>
    …
</select>
```

Note that if you want to catch on the server mutiple selected items, you have to pass them ass array by adding the "[]" atthe end of the name.

… and some more. All of these objects have to be placed inside a HTML form.

Example

```
<form action="some_file.php" method="post">
     Enter your name: <input type="text" name="DATA_name"><br>
     <input type="submit" name="BUTTON_go" value="Go">
</form>
```

Most of the time, we don't want to have two scripts, so the script is calling itself. In PHP, the most elegant way to achieve this, is shown in the next example:

$_SERVER is a special array which holds different server related values. The field "SCRIPT_NAME" contains the name of the actual called script.

```
<form action="<?=$_SERVER['SCRIPT_NAME']?>" method="post">
     Enter your name: <input type="text" name="DATA_name"><br>
     <input type="submit" name="BUTTON_go" value="Go">
</form>
```

By this way, the script can test if it is called from a form or not an created the correct output accordingly to this test:

```
<?php
     if(isset($_POST['BUTTON_go']))
     {
          // do calculations and output result
     }
     else
     {    // output the HTML form (see code above)
?>
          <!-- the HTML form -->
<?php
     }
?>
```

Practice

Modify all exercises of the chapter 3.1. by adding an HTML form and thus making them more user friendly!

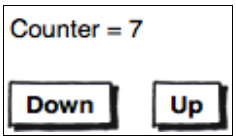
3.4. Hidden data

When jumping from one web page to another, we often need to transmit data between them but we do not want it to be visible to the use. In these cases we use hidden fields. They work exactly like standard input text fields, but do not show up on the page.

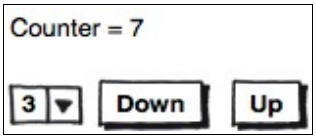
```
<input type="hidden" name="DATA_myHiddenField" value="myValue">
```

Practice

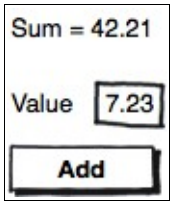
1. Write a script with the above given HTML example line and display the **$_POST** variable after having send it back to itself. Watch the output closely!
2. Write a page that allow to count things:



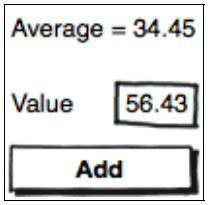
   A click op the button "Down" decrements the counter and a click on the button "Up" increments it by one unit.
3. Make a copy of the previous exercise and modify it so that the user can select the step from a drop-down list holding the values 1 to 7.



4. Write a page that allow the user to sum up different numbers:



5. Write a page that allow the user to calculate the average of subsequent entered numbers:



3.5. Posting arrays

In the previous chapter, we saw that it is quite easy to post textfields, but what happens if we want to post data that belongs together or is always of the same type so that we could capture it immediately as array in our script?

Output the debug message only if the button has been pressed.

Example

```
<?php
     if(isset($_POST['BUTTON_submit']))
     {
          echo '<pre>'.print_r($_POST,true).'</pre>';
     }
?>
<form action="<?=$_SERVER['SCRIPT_NAME']?>" method="post">
     Enter 5 numbers:<br>
     <input type="text" name="DATA_numbers[]"><br>
     <input type="text" name="DATA_numbers[]"><br>
     <input type="text" name="DATA_numbers[]"><br>
     <input type="text" name="DATA_numbers[]"><br>
     <input type="text" name="DATA_numbers[]"><br>
     <button name="BUTTON_submit">Submit</button>
</form>
```

Once posted, the received data is being displayed like this (for the number 23, 34, 43, 23 and 12 entered into the 5 fields):

```
Array
(
    [DATA_numbers] => Array
        (
            [0] => 23
            [1] => 34
            [2] => 42
            [3] => 23
            [4] => 12
        )

    [BUTTON_submit] =>
)
```

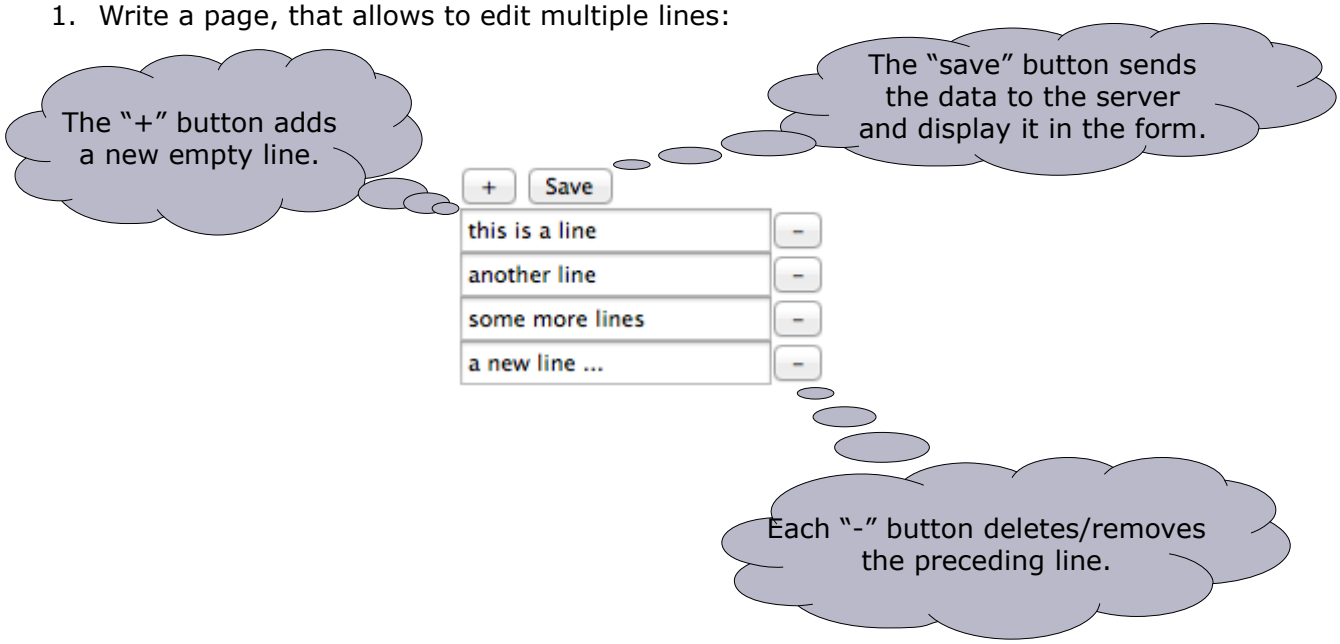
As you can see, the post variable contains an array, which our script could immediately use to work with.

By the way, did you know that in PHP you can add an entry to the end of an array as simple as like this?

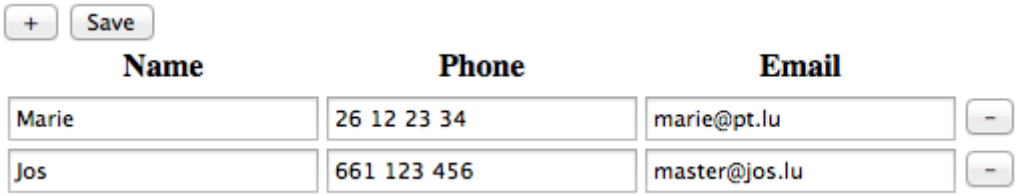
```
$someArray[] = "A new entry";
```

Practice

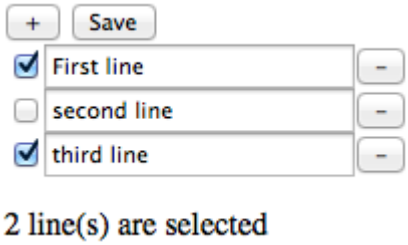
1. Write a page, that allows to edit multiple lines:



2. Write a page, that allows to manage a phonebook. Each person has to be an array with the following fields:
   - name
   - phone
   - email



3. Write a page with selectable lines. At the bottom of the page, the number of selected lines has to be indicated.



4. The following array is given:

```
$fishes = array(
  'Goldfish'    => 2,
  'Guppy'       => 13,
  'Neon Tetra' => 8
);
```

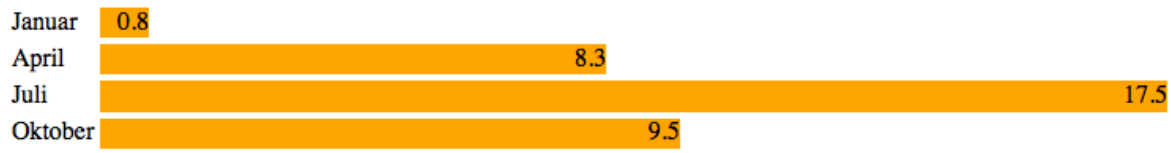
Write a script, that generates a table as the one represented below in which there is a column for every kind of fish and in every column there are as many rows as fishes indicated in the array.

Goldfish	Guppy	Neon Tetra
1)	1)	1)
2)	2)	2)
	3)	3)
	4)	4)
	5)	5)
	6)	6)
	7)	7)
	8)	8)
	9)	
	10)	
	11)	
	12)	
	13)	

5. Write a page that contains a counter as well as the following two links:
   - count up to increment the counter by one unit
   - count down to decrement the counter by one unit
6. Make a copy of the previous exercise, replacing the links with buttons. What problem are you running into? How could you solve it?
7. The following array is given:

```
$temperatures = array(
     'Januar'  => 0.8,
     'April'   => 8.3,
     'Juli'    => 17.5,
     'Oktober' => 9.5
);
```

Now write a script that generate the following bar chart:



Also pay attention if the array contains negative values!

3.6. Flattening arrays

In many situations there is need to transmit the data of a simple non-associative array from one page to another. This can be easily achieved using the two functions **implode(...)** and **explode(...)**.

3.6.1. Imploding an array

This means that the array is being flattened down to a simple symbol separated string.

Example

```
$myCars=array("Saab","Volvo","BMW");
echo implode(",",$myCars);
echo implode("|",$myCars);
```

The first parameter is the symbol (this may be more than one!) used to glue together the different elements of the array.

will display these two lines

```
Saab,Volvo,BMW
Saab|Volvo|BMW
```

Using the **implode** function it is thus quite easy to transmit an entire simple non-associative array from one page to another using a hidden field in a form.

3.6.2. Exploding an array

This means that the array is being flattened down to a simple symbol separated string.

Example

```
$myString="Saab,Volvo,BMW";
$myCars=explode(",",$myString);
print_r($myCars);
```

The first parameter is the symbol (this may be more than one!) used to glue together the different elements of the array.

will display these two lines

```
Array
(
    [0] => "Saab",
    [1] => "Volvo",
    [2] => "BMW"
);
```

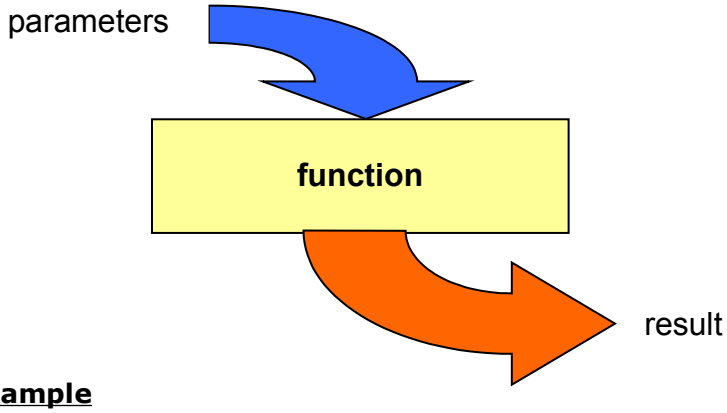
Using the **explode** function we can thus reconstruct an array from a given hidden field transmitted by another page.

You will have to use these function in the upcoming practice section of the next chapter.

4. Functions

As for Javascript, developers can define their own self-made functions in PHP. This even works the same way as in Javascript:

A function, also called sub-routine, is a small, mostly independent, program. The developer can pass data to the function by specifying inputs. Most functions do something with these inputs and then return a result. A function always follows the IPO[2] method. The inputs of a function are called **parameters**.



Example

```
function getSum($a,$b)  // input
{
     $sum = $a + $b;    // processing
     return $sum;       // output
}
```

This function has two parameters. They are here called "$a" and "$b".

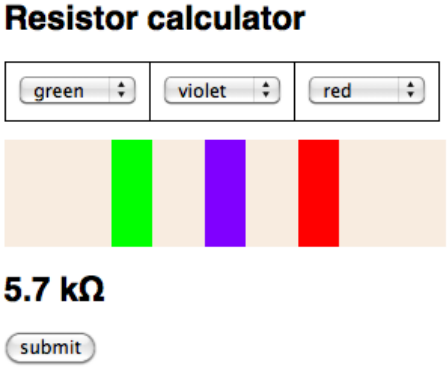
Using the parameters, we can "give" the function some values. Here is an example of how our self-created function could be tested:

```
echo getSum(3,8);
```

You see, functions are nothing new to you. You already used them since your very first steps in JavaScript. What is new, is that you are now able to write your own functions!

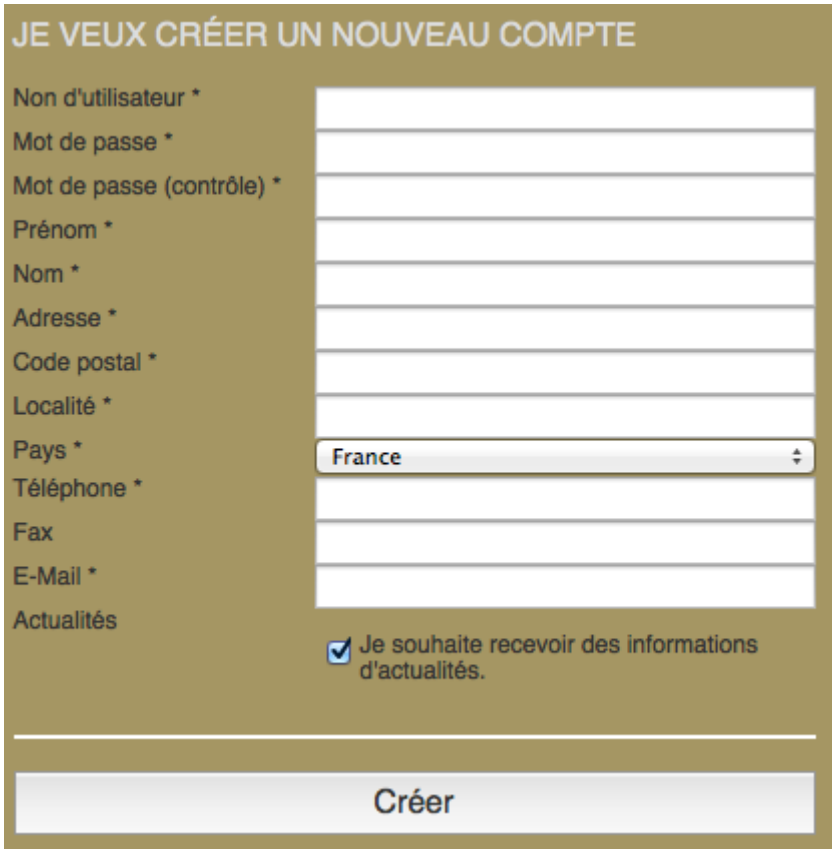
Practice

1. Write, then use a function to calculate the needed iterations of the process described in practice 3.1.4.
2. Write a resistor calculator!



2 input, processing, output

3. Create an inscription page like this one:



The form has to be shown up, pre-filled with the data entered by the user, as long as the following constraints are not respected:

- All requires fields (*) must have a value.
- The password must match the password control.
- The username must meet the following rules:
  - minimum 6 characters
  - only a-z, A-Z and 0-9 allowed
- The email address must be valid.

The list of countries will be supplied to you in form of a text file. To read it, use the function **`file_get_contents(...)`**. Write yourself a couple of useful functions to check the correctness of the different fields.

4. Write a page that display 8 bulbs. A clic on a bulb toggles the state of that bulb. For each of the buttons, write a separate PHP function to accomplish the given works:

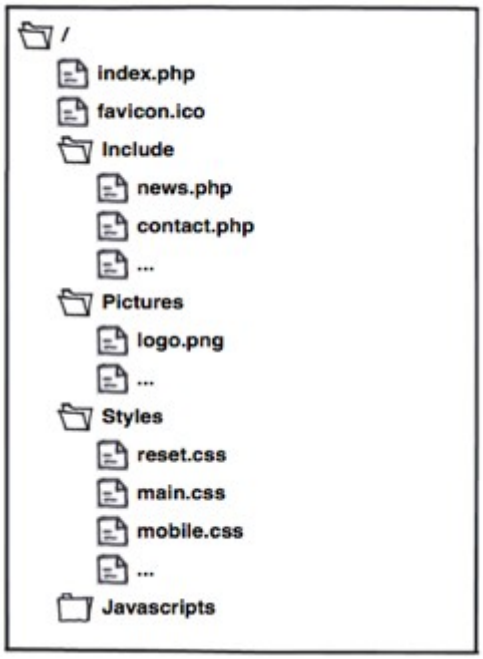
5. Using the previous exercise, write the following game:
   - The script determines a random number of the interval [0,255] and displays it in binary form using 8 bulbs.
   - The user has to determine the corresponding decimal value and enter it in an edit field.
   - The scripts counts how many attempts the user had to make before he enters the right value.

5. Well structured sites

Developing well structured and easy tom maintainable web sites isn't that complicated if you stick to some very basics rules. Probably you'll find dozens of good structures on the net, which do certainly have all their very own advantages and disadvantages. The here presented structure may not be the best, but it's a fairly good one to start with.

5.1. The physical structure

First of all we need to organise our files. This will be done like this:

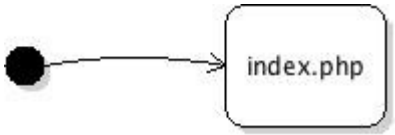


- The root directory only contains the **`index.php`** file. This is our main file and all links will point to it.
- The **`Include`** directory holds all files to be included by the **`index.php`**. Depending on the parameters (GET or POST) passed, the **`index.php`** file choses what content to include.
- All pictures we use on our site are going to be put into the **`Pictures`** folder.
- The **`Styles`** folder will be populated with the sites CSS files.
- Finally all JS files that are to be deployed are being copied into the **`Javascripts`** folder.

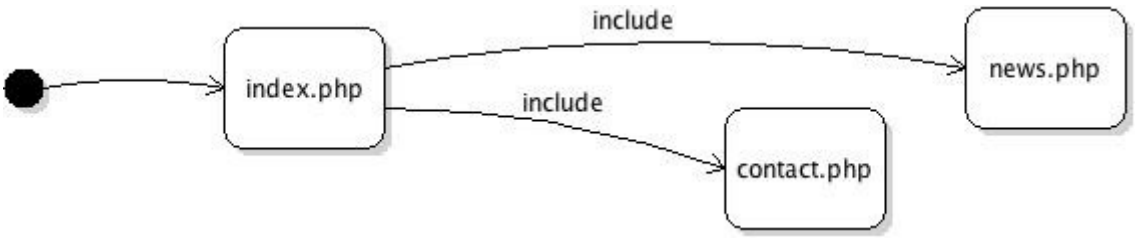
5.2. The logical structure

Besides the physical structure, meaning the files and the folders, it is important to determine the logical structure. This is how the files work together.

As already told, each link from the entire site has to go to the **`index.php`** file, of even better, immediately to **`$_SERVER['SCRIPT_NAME']`** as this allows us to use another filename as **`index.php`** and thus is more flexible.



Once a visitor arrives at the **index.php** file, the values of the parameters (GET or POST) passed to the file are used to decide what content page to include and display to the visitor:



5.3. Advantages

1. All layout related things, like header, footer, menu or others, are kept in the **index.php** file.
2. In case we need to add a page, we only need to add the required PHP file in the **Include** directory and allow it to be loaded by the **index.php**.
3. As all links point to the **index.php** file and this one is our main entry point to the web site, it is easy to implement all kind of securities or surfing enhancements.

Practice

1. Create a web site which displays the content of the pages 6, 8 and 12 from the following PDF document:

   http://www.ltam.lu/Files/CloseUp/CloseUp2013_p029_061.pdf

2. Do you have any interesting hobbies? Chose one and create a website of it. Don't forget to stick to the above given structure while implementing it!

   Here are some examples you may want to take a look at:

   - http://arduino.fisch.lu
   - http://mindstorms.computerclub.lu (outdated, but that's of no importance ...)
   - http://www.levygraphie.lu

6. Advanced features

6.1. Reading files

In many cases web developers need to read or write data from, respectively to files. In PHP there are many ways to achieve this. When dealing with textual data, the functions **`file_get_contents(...)`** and **`file_put_contents(...)`** are certainly among the easiest.

6.1.1. Reading from files

Reading data from a file is quite easy:

Example

```
$file = file_get_contents('cars.txt');
```

Now the variable **`$file`** will contain the entire content of the file 'cars.txt'.

6.1.2. Writing to files

This is as easy as reading data from a file:

Example

```
$cars = "Volvo\nRenault\nJaguar";
file_put_contents('cars.txt', $cars);
```

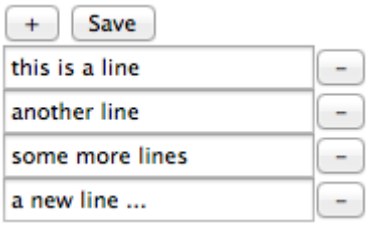
6.1.3. Testing files

If you want to check for the existence of the a given file, you my want to use the function **`file_exists(...)`**.

```
if (file_exists("cars.txt"))
{
    // do something
}
```

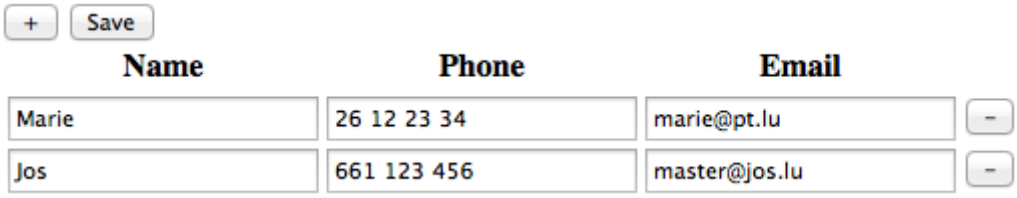
Practice

1. Write a page that contains a hit counter!
2. Write a page, that allows to edit multiple lines:



   The lines are to be store in a file called "2.txt", which is being loaded automatically on each page request.

3. Write a page, that allows to manage a phonebook. Each person has to be an array with the following fields:
   - name
   - phone
   - email



   The persons are to be stored in a single file called "3.txt". Each line of the file has to be a person. The file has to be loaded automatically upon each request to the given page.

4. Implement a very simple guestbook. Each entry in the guestbook has 2 fields:
   - The name of the person and
   - the message it left.

   The upper part of the guestbook allows a user to enter his name and message. The lower part displays all guestbook entries in reversed order.

6.2. Character encoding

A major problem to deal with is text encoding. A character encoding system consists of a code that pairs each character to a given bit pattern. Actually in our region we mostly work with "UTF-8" and "ISO-8859-1".

The default character encoding to be used is nowadays "UTF-8", but other data source may not used it. In that case it is good to know that PHP offers some useful function to encode and decode text to and from "UTF-8".

6.2.1. Decoding

Decoding a string works like this:

Example

```
$nonUtf8String = utf8_decode($utf8String);
```

6.2.2. Encoding

Encoding a string is simply the inverse:

Example

```
$utf8String = utf8_encode($nonUtf8String);
```

6.2.3. HTML encoding

In HTML a lot of symbols, like the famous ">" have to be converted to HTML entities. The function **`htmlentities(...)`** does a great job, but only since PHP 5.4.0. it uses "UTF-8" as default encoding. Version before that have to specify it manually.

```
$htmlString = htmlentities($string,ENT_QUOTES,"UTF-8");
```

7. Sessions

Most of the time when writing useful web applications, we, the developer, have an urgent need to "remember" things inside our scripts. Until now, this document covered the following ways to archive this:

- pasting the needed information from page to page using "hidden fields", which generates an overhead of network traffic an needs a lot of coding,
- writing the information to a file, which reveals the problem that we do not know the time when this temporary file could be deleted again.

Fortunately there is a much easier way PHP offers to make scripts "remember" information on the server. The technique is called "sessions":

1. For each user, a session file is being created on the server. It is being deleted automatically, thus the developer has not to deal with that problem.
2. We, the developer, can tell the script what information (simple or complex) to store an we can access it later in another script using the same session information.
3. Closing the browser closes also the session.
4. Using different browsers on the same computer will create a session for each browser.

OK, this is quite abstract, to let's start with a simple example.

Example:

```
// start the session
session_start();
// init counter if needed
if(!isset($_SESSION['counter']))
{
     $_SESSION['counter']=0;
}
// increment counter
$_SESSION['counter']++;
// output counter
echo $_SESSION['counter'];
```

This command initializes the session variable. It is only available and stores information AFTER this call.

Yes, the $_SESSION is an array and can thus hold any type of data, even other arrays!

Other important session related commands are:

- session_destroy() Destroys all data registered to a session
- session_write_close () Write session data and end session

For any further information about session, please stick to the official web site:

http://www.php.net/manual/en/book.session.php

Practice

1. Try the above given counter example and find answers to the following questions:
    1. What happens if you close an re-open your browser?
    1. What happens if you open the same page in another browser?
    2. What happens if you open the page on another computer?
2. Write the game "Tic-Tac-Toe" using the following page-flow:
    3. ask both users for their names,
    4. display the field as long as none of them has won and it is not full,
    5. display the result
    6. goto #1
3. Write a password protected page (a single file!). Once the password is entered, the hidden treasure is being displayed, then try the following:
    7. Enter the password and see the treasure. Surf to another page and come back to the password protected page. You should see immediately the treasure again.
    8. Enter the password and see the treasure. Close the browser and re-open it, then load again the password protected page. You should be asked for the password.
    9. Enter the password and see the treasure. Open the password protected page in another browser on your computer. You should be asked for the password.
4. Write a page (a single file!) which allows you to fill up a shopping basket. The items sold in this mini-shop are the following:
    10. bananas 2.50 €
    11. milk 1.25 €
    12. crashed ice 0.45 €
    13. plastic cup 1.35 €

    The user should be able to
    - add or remove a single item from the basket using a (+) or (-) button
    - see how many of each items are in the basket
    - see the total value of his basket

5. Survey generator

   A "survey" has a title and consists of a a set of different questions. Each question has a single line text (the question it self), a type (text, radio, checkbox, textarea) and up to 10 possible responses (ignored for the types "text" and "textarea").

   1. Create a script that allows the user to create a survey and define a certain number of questions. This is what is called the "edit mode".
      1. The user must be able to set the title of the survey.
      2. The user must be able to insert a question before or after an existing question.
      3. The user must be able to edit a question.
      4. The user must be able to delete a question.
   2. Display and test the survey. This is what is called the "display mode".
      - The user must be able to switch at any moment between the edit and the display mode.
   3. Export the survey as text file, called "<title>.surv" using the following format:
      - Line 1 = the title of the survey
      - Line 2 = the number of questions
      - Line 3 = the text of the first first question
      - Line 4 = the type of the first question
      - Line 5 – 15 = the 10 possible answers
      - Line 16 = the text of the second questions
      - Line 17 = …
      - …
   4. Import the survey from a text file, which uses the above format.

6.

8. MySQL database binding

There are different methods to access a MySQL database in PHP. The here described method uses the "mysqli" PHP extension and we are going to use it's procedural interface.

```
// open a link to the database
$link = mysqli_connect("127.0.0.1", "username", "password", "database");

// test if the connection is open, exit otherwise
if (mysqli_connect_errno())
{
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

// define the query
$query = "SELECT * FROM personnes ODER BY name";
//echo $query;

// if we got a result
if ($result = mysqli_query($link, $query))
{
     // loop through each row
    for($i=0; $i<mysqli_num_rows($result); $i++)
    {
        // get the row
        $row = mysqli_fetch_assoc($result);
        // output the name
        echo $row["name"].'<br>';
    }
    // free up the result
    mysqli_free_result($result);
}
// close the connection
mysqli_close($link);
```

Explanation

- First of all, a connection to a given database is opened.
- Next, we test that the connection has been opened. This step may be optional in some cases.
- Before executing a query, it is a good idea to write it in a separate variable as this allows us easy debugging.
- Next we are going to retrieve the result set an test at the same time if we have a valid result. In that case,
  - we loop through the result set, retrieving each line and outputting a given field.
- At the end, the result set is freed up and the connection is closed.

Practice

For each of the following practices, create in your database this table and fill it up with a lot of data:

persons
idPerson name firstname age tel email

1. Write a page that displays the name of each person in an ordered list.
2. Write a page that displays all the persons in an HTML table. The table data must be sorted by the name in ascending order.
3. Modify the previous exercise by adding buttons to the header row of your table allowing you to sort the data by any field in ascending order.
4. Modify the previous exercise by adding a second button to the header row of the table which allows to sort the data by the given field in descending order.
5.