1 Introduction.

Beyond the cosmetic part of organising web-pages, we must also think about how web-pages interact with users, how data about different users can be stored in the server. This is important for many applications, from advertising to banking. It can be achieved mainly through server side programming, i.e. programs that are executed on the server - unlike HTML, which is parsed by the client.

Server-side programming

HTML and CSS are methods by which we can specify how our web pages are formatted. However, this a one-sided interaction. The client, such as Firefox, Chrome, Internet Explorer, or Safari, asks for the web page, gets the data, and renders it.

HTML and CSS: client-side execution

1. Client sends page request. This happens through the HTTP protocol.
2. Server responds with page. This is sent over via HTTP again.
3. Client renders page. If the webpage is a single HTML file, then this step is immediate. However, if the page includes external files such as images or CSS files, then these have to be retrieved via further HTTP requests.

Need for server-side programs

Many times we wish to execute programs also on the server side. This is necessary for many applications such as.

1. Web forums.
2. Online banking.
3. Shopping websites.
4. Any site requiring user data.

Server-side programming

Server side programming can be accomplished with many languages, but the most common one is PHP. In any such program, when the user requests

a page, the server *does not* simply return the page to the user. Rather, it first *executes* the program requested, and it returns its *output*.

Inputs to programs
The server usually needs some input to the program it is going to execute. Sometimes this can be achieved by specifying different URLs, but it is most common to do it with the POST method and HTML forms.

Storage
While the program usually just needs to execute something and return an HTML page, sometimes we actually need to store some data. This data can be stored in many different ways: via files or databases on the server, via *cookies* stored in the client, or via *session* files stored in the server.

HTML Forms

User input can be obtained both directly and indirectly. Forms are used to take user input directly. Here is a simple example.

Example 1. The following HTML constructs a form to obtain user data, and sends it to a PHP script

```
1 <form action="script.php">
  Name: <input type="text" name="name">
3 <br>
  Surname: <input type="text" name="surname">
5 <br>
  <input type="submit" value="Submit">
7 </form>
```

What can we *do* with the user input? Perhaps in this case we'd like to save the input in some database. However, the client-server model does not seem to allow us this option. *Forms can be used to call and pass data programs running on the server.* These are called server-side scripts. In the example above, we would run a script called `script.php` on the server. This would also pass the value of name and surname we have inputted in the form to the script. That way, our HTML can communicate with any script.

The action method

There are two ways to send data using forms. The first uses the GET method and the second the POST method of the HTTP protocol.

GET method

```
1 <form action="do_something.php" method="get">
```

Forms the URL `do_something.php?firstname=X&Lastname=Y`

- When server state should not be altered.
- Unsafe.

POST method

```
1 <form action="do_something.php" method="post">
```

Does not send the data through the URL.

- When server state should be altered.
- Safer.

PHP (PHP: Hypertext Preprocessor)

PHP is a complete language, which we can use to write programs that are going to be executed on the server. It can take user input from HTML forms, communicate with other programs on the server (such as databases). It *always* generates a web-page as a side-effect.

PHP files

- Can contain text, HTML, CSS, Javascript, PHP code.
- PHP is executed on the server.
- PHP files have the `.php` extension.

Main uses

- Data processing.
- Dynamic page rendering.

Running and debugging PHP

On the server

To run PHP on the server, you need to have installed PHP and your web server's (e.g. Apache) PHP extension and correctly configured them Your university should have provided you with a preconfigured directory where you can put your HTML and PHP scripts. You can also use your own computer as the server, but then you'd have to set it up yourself.

Locally, without a server

Running PHP locally is the easiest way. It also gives you important debugging information which you can't otherwise see from your environment. To run PHP locally, open a shell (terminal) and write

```
php5 myCode.php
```

This does not invoke a server, so you must take care that everything connects properly, especially input from HTML web forms.

Basic PHP code

PHP can be embedded within HTML. In fact, its most direct use is to just *output* HTML. To see what this means, execute the following script using:

```
php5 myScript.php
```

assuming you have named the script myScript.php.

myScript.php

```
<!DOCTYPE html>
<html>
<body>

<h1>My first PHP page</h1>

<?php
```

```
  echo "Hello World!";
9 ?>

11 </body>
  </html>
```

`echo` is a command that just outputs its argument. You should observe that the output of this script is this:

Output of myScript.php

```
  <!DOCTYPE html>
2 <html>
  <body>
4
  <h1>My first PHP page</h1>
6
  Hello World!
8
  </body>
10 </html>
```

The PHP code has disappeared and has been replaced by its output.

Otherwise, PHP is just like another language, with comments, conditional execution, loops, and variables.

Comments

```
  // Single-line comment
2 # single-line comment
  /* multi-line
4    comment */
```

Keywords

```
echo "Hello world!<br>";
```

Keywords are case insensitive. That means that `echo` is the same as `Echo` or `ECHO`.

Variables

```
$name = "Jack";
echo "Hello " . $name . "<br>";
```

Variables are case sensitive. That means that `name` is different from `Name` or `NAME`.

Conditional statements

Sometimes it's convenient to have conditional statements.

```
$t = date("H"); // get the current hour of the day
if ($t < "10") {
  echo "Good morning!";
} elseif ($t > "20") {
  echo "Good evening!";
} else {
  echo "Good day!";
}
```

Including content

Sometimes it's useful to have parts of web pages in separate files

```
<html>
<body>
<h1>Welcome to my home page!</h1>
<p>Some text.</p>
<p>Some more text.</p>
<?php include 'footer.php';?>
</body>
</html>
```

2 Scope

Scope is extremely important in PHP. While there is the standard idea of variables being normally only accessible where they are declared, it has some additional complexity: there are special variable names that can be associated to specific browsers and browser sessions. These provide an easy method to have user-specific data.

Local vs global scope

```
$x = 5; // global scope

function myTest() {
    // using x inside this function will generate an error
    echo "<p>Variable x inside function is: $x</p>";
}
myTest();

echo "<p>Variable x outside function is: $x</p>";
```

A variable with global scope is only visible in global scope

Scope

```
function myTest() {
    // using x inside this function will work, as it is declared here
    $x = 5; // local  scope
    echo "<p>Variable x inside function is: $x</p>";
}
myTest();

// using x inside this function will fail, as it is not declared here
echo "<p>Variable x outside function is: $x</p>";
```

A variable with locale scope is only visible in local scope

Scope

```
$x = 1; // global scope

function myTest() {
    $x = 5; // local  scope
```

```
5     echo "<p>Variable x inside function is: $x</p>";
  }
7 myTest();

9 // using x inside this function will fail, as it is
     not declared here
  echo "<p>Variable x outside function is: $x</p>";
```

As a matter of fact, variables only exist at the scope in which they are declared. You can think of the same variable declared in different scopes a completely different variable.

Exercise 1. What is the value of the variable inside and outside the function?

Global variables

Global variables are special. They are not only available in the global scope. They can be made available in local scopes in two ways. Via the `global` keyword and via the `globals` array

Accessing global variables

```
  $AGlobalVariable = 1;
2 function demonstration($Argument) {
      global $AGlobalVariable;
4     $AGlobalVariable = $Argument;
      echo "$AGlobalVariable";
6 }
  demonstration(10);
8 echo $AGobalVariable;
```

Global variables can also be accessed via `$GLOBALS["AGlobalVariable"]`

Tutorial

Follow the tutorial at `http://www.w3schools.com/php/` until PHP Echo / Print.

There are many other types of special variables, such as `$_GET`, `$_POST`, `$_SESSION` and `$_COOKIE`. These are used for obtaining, storing and accessing user and browser data.

3 Using HTML forms to pass data to PHP.

Processing form data using PHP

Since we can send data over HTTP with at least two different methods, PHP has two different ways to collect it. In both cases, the data is store in a specially named variable.

GET data

```
echo $_GET["name"]; // Print the form variable
    called 'name'
```

You should use GET for data obtained through an HTML form with a *get* method.

POST data

```
1 echo $_POST["name"]; // Print the form variable
    called 'name'
```

You should use POST to access data obtained through an HTML form with a *post* method.

Safety and security of GET

GET is not safe, as the data is sent as part of the URL. Consequently

- Anybody can see the data.
- There is a limit to the amount of data that can be sent.

With POST, we can use an HTTPS connection to send the data encrypted.

3.1 HTTPS

HTTPS is the secure version of HTTP. Using it correctly ensures that nobody can see what data the user requests nor the web pages he obtains. However, they can still see which server the user connects to. This is most commonly used for sending over usernames, passwords, and other sensitive data; but more generally for privacy.

HTTPS

HTTP Secure

- For every session (connection), we connect to the host using TLS - a secure connection.
- We then transfer data using HTTP as normal, but this data is encrypted.

TLS

- The client connects to the server.
- They perform a "handshake" during which they agree on a secret key.
- Then all communication is encrypted with this secret key.

The secret handshake

Alice and Bob wish to share a *secret key*. Let us say that they agree on a *public* numbers q first. Then the idea is that

- Alice chooses a secret x and sends $A = q^x$ to Bob.
- Bob chooses a secret y and sends $B = q^y$ to Alice.

Now both can calculate the following *shared secret key*

$$A^y = (q^x)^y = q^{xy} = B^x.$$

Technical details

Usually, this is done in $\mathbb{Z}_p^*$, where p is prime. This makes it hard to find x, y from A, B, q.

Exercise 2. Make a webpage called `conditional.html` with a form using the POST method, sending data to a separate script called `conditional.php`.

The webpage should ask the user his age, and if it's lower than 18, print "Access denied". Otherwise, it should show some content using the PHP `include` keyword. This should point to a file containing *the body* of your HTML content.

You should store the webpage files in your local directory: `~login/www/`

They then become accessible at `https://hebergement-peda.univ-lille3.fr/~login`

Solution. This is easy to solve. First, we create a simple web page that contains a header, the form and a link to the script. Everything else is done by the script.

```
<html>
  <head>
  <link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
   <body>
      <form action="conditional.php" method="POST">
        Age: <input type="text" name="age" />
        <input type="submit" />
      </form>
   </body>
</html>
```

pages/conditional.html

The script is quite simple. It reads the form data, and if the condition is satisfied, it reads in the HTML content.

```
<?php
echo"testing";
if($_POST["age"] >= 18) {
    include("paragraphs.html");
} else {
    echo "Access denied";
}
?>
```

pages/conditional.php

The HTML content itself is well-formed HTML, containing everything we normally find within the `body` tags. This is my own example:

```
  <h1>Heading</h1>
<p class="fixed">Text</p>
  <ul>
  <li>List</li>
  <li id="important">First</li>
  <li class="latin">Second</li>
  </ul>
  <p>This is a paragraph in English.</p>
  <p class="latin">Thisu isu a paragraphu in fakeum Latinum</p>
```

```
11  <p>More text in a paragraph. AS ewoi weroinn wrwoir
     n roiwen oi rwoiij woij vowij  rwej Lorem ipsum</p>

13   <p>More text in a paragraph. AS ewoi weroinn
    wrwoir n roiwen oi rwoiij woij vowij  rwej Lorem
    ipsum</p>

15   <p>More text in a paragraph. AS ewoi weroinn
    wrwoir n roiwen oi rwoiij woij vowij  rwej Lorem
    ipsum</p>

17     <p>More text in a paragraph. AS ewoi weroinn
    wrwoir n roiwen oi rwoiij woij vowij  rwej Lorem
    ipsum</p>

19     <p>More text in a paragraph. AS ewoi weroinn
    wrwoir n roiwen oi rwoiij woij vowij  rwej Lorem
    ipsum</p>

21       <p>More text in a paragraph. AS ewoi weroinn
    wrwoir n roiwen oi rwoiij woij vowij  rwej Lorem
    ipsum</p>
```

pages/paragraphs.html

□

4 Cookies

Cookies

- Cookies are stored in the *user's computer*. We use the HTTP protocol to request a cookie to be stored. The user is free to decline acceptance of a cookie. Try and disable cookies in your web browser to see how often you get these requests.
- Cookies can be used for
  1. Session management.
  2. Personalization.
  3. Tracking.

"HTTP cookie exchange" by Tizio - Own work. Licensed under CC BY-SA 3.0 via Commons https://commons.wikimedia.org/wiki/File:HTTP_cookie_exchange.svg#/media/File:HTTP_cookie_exchange.svg

Cookies in PHP

```
setcookie(name, value, expire, path, domain, secure,
    httponly);
```

- name: the name of the cookie. The only necessary argument.
- value: the value of the cookie.
- expire: when the cookie expires.
  - If 0, the cookie expires at *session end*. When the browser closes.
  - If non-zero, it ends at the specified time. Use `time() + nSeconds` to expire nSeconds after the current time
- domain: where the cookie is visible.
- secure: if TRUE, only transmit the cookie over HTTPS.
- httponly: if TRUE only available through HTTP.

Even though secure, httponly, are FALSE by default, it's best if they are set to TRUE for security reasons.

Example 2. The following code sets up a cookie with the name user, value John Doe, and set to expire in 30 seconds. It is also available on all the webpages in the server's domain. The second part of the code checks if the cookie has been set.

```
1
  <?php
3 $name = "user";
  $value = "John Doe";
5 setcookie($name, $value, time() + 30, "/");
  ?>
7 <html>
  <body>
9 <?php
  if(!isset($_COOKIE[$name])) {
11     echo "Cookie  '" . $name . "'  not set!";
  } else {
13     echo "Cookie '" . $name . "' is set!<br>";
       echo "Value is: " . $_COOKIE[$name];
15 }
  ?>
17 </body>
  </html>
```

The first time you run this example through your web browser, the cookie will not have been already set. The second time the cookie should already exist. However, executing it directly from the command line will not work, as it's the web server and client's job to exchange the cookies through HTTP.

It is important to set the variables in before the HTML part. That way they are available everywhere.

Exercise 3. Continue the previous example, with the web-page asking a user their age. This time, you are going to store a cookie, instead. Implement the following:

- If the user hasn't got a cookie yet, ask the user's age, and store it in a cookie with name "age"; the cookie should expire at the end of the session.
- If the user already has a cookie, then set the age variable according to that.
- In either case, show "access denied" if the user age is less than 18, and the complete HTML body if they are older.

You should store the webpage files in your local directory: `~login/www/` – they then become accessible at `https://hebergement-peda.univ-lille3.fr/~login`

If you have problems with this, you need to contact the system administrators.

solution. This is easy to solve. First, we create a web page that tests if the cookie is there, and if not, it includes a form

```php
<?php
## setcookie("age", -1, 0, "/");
?>
<html>
<body>
<?php

## Ask for a new cookie if the age was not already set.
if(!isset($_COOKIE["age"])) {
    echo "I haven't seen you before. Please enter your age <br>";
    include "ask_age.html";
} else {
    ## Check if the cookie is there.
    if ($_COOKIE["age"] < 18) {
        echo "Access denied";
    } else {
        echo "Access granted";
    }
    echo "A bad way to provide access control.";
}

?>
</body>
</html>
```

pages/cookie.php

The form is very simple. It asks the age and then calls another script, `age.php`, which stores it in the cookie.

```php
<form action="age.php" method="POST">
  Age: <input type="text" name="age" />
  <input type="submit" />
</form>
```

pages/ask_age.html

The final script just stores the value obtained by POST to the cookie, and then reloads the original page.

```php
<?php
echo "setting cookie";
setcookie("age", $_POST["age"]);
header("Location: cookie.php");
?>
```

pages/age.php

□

5 Sessions

Another way to store data associated with specific clients is with sessions. These are always stored in the special `$_SESSION` variable. It is important to understand that *multiple copies* of these variables may exist. This is because each session is associated with a specific client connection.

Sesssion

- Alternative way to store information across many web pages.
- Information is stored *in the server* and not on the user's computer.
- Information remains in a web-page.

Example 3 (Setting session variables)**.** The most basic thing we can do with session variables is to set them.

```
<?php
// Start the session
session_start(); // must be in the beginning
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>

</body>
</html>
```

Example 4 (Reading session variables)**.** Of course, once set, a session variable can also be read.

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

```

```
<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>

</body>
</html>
```

Exercise 4. As a small exercise, open *two private browser windows*. In the first window, run the script of Example 3, and then the script of Example 4.

In the second window, run only the script of Example 4.

You should observe that in the second window, the session variables are not available. This is because the session variables only exist in the browser session where they were declared.

Session management

`session_start()`
Begins a new session

`session_unset()`
Clear all the variables from the current session. This clears all the variables, but does not affect the session storage.

`session_destroy()`
Destroy the current session. This deletes the session file in the server file system. It also closes the session, and you must start a new session afterwards.

`session_name($NAME)`
Create a named session. Normally, session names are randomly generated so that each connection has a unique name. If we set a specific name for the session, this allows us to *share data* between connections. It is best to use this together with `session_write_close()` if necessary.

Accessing session variables

```
$_SESSION[$name]
```

6 Persistent data

Frequently we need to store data in a persistent way on the server. The simplest approach is to use named sessions, but this has many disadvantages: the session variables may not exist after a restart, while it may not be possible to use with many users at the same time. A more permanent method is to use files, but this also suffers from lack of thread safety. The standard approach is to use a database, such as mySQL or sqlite, for which there is very good support in PHP.