Server side programming - PHP

Dr. Amir Djouama

amir.djouama@ensia.edu.dz

Introduction (1)

- PHP: Personal Home Pages
- Officially: PHP Hypertext Preprocessor
- Programming language available for different operating systems.
- Open Source software: Royalty free.

Introduction (2)

- Server-side scripting language.
- Interpreted language.
- We say: PHP scripts or PHP programs
- Allows you to generate HTML code and send it to the client.

Features

- Very close to the C language
- Integrates into the heart of an HTML page
- PHP instructions do not appear in the final result.
- No way to access source code from client.
- Other languages: We write the program to display (output) an HTML page.
- PHP: We write an HTML page with included code.

What can PHP do?

- Collect data
- Dynamically generates web pages
- Send / Receive cookies.
- It supports services using protocols: IMAP, SNMP, POP3, HTTP, …
- Interact with other protocols by opening connections to various TCP/IP ports
- It has an important set of tools: image manipulation and generation, file processing, PDF creation, …

What can PHP do?

- Supports a large number of DBMS:
  - mysql
  - dBase
  - FilePro
  - Informix
  - Interbase
  - mSQL
  - ODBC
  - Oracle
  - Unix dbm, …

Advantages (1)

- Facilitates the development of dynamic and interactive websites.
- Simple and clear answer.
- The task of interpreting commands delegated to a component called engine.
- The engine is installed on the server
- Rich library: writing complex algorithms in a few lines.
- Function for splitting character strings.
- Give system info

Advantages (2)

- Easy maintenance / reusable
- Flexible language / classic and practical syntax.
  - Instructions known.
  - Based on standards.
  - Usage shortcuts.
  - Unnecessary declaration of variables.
  - Conversion system provided by the PHP engine.

Inserting PHP code

- We insert PHP code in full HTML page:
  - `<?PHP …?>`
- The page must have the extension `.php`

Interpretation of PHP code

- Any block of this kind will be interpreted then an output HTML code (according to instr .) is generated instead of the instructions then sent to the client.
- Any other block is sent untouched to the client.
- In the event of errors (lexical or syntactic), a message is integrated into the exit code and execution interrupted (according to the configuration of php.ini).

Minimal PHP

- Output: `echo statement`
- Used to output HTML code to the page.
- Example :
  - `echo "character string";`
- PHP is case sensitive.

Example

- Server side

```
<HTML>
<HEAD>
<TITLE> test </TITLE>
</HEAD>
<BODY>
<?php echo("2nd Y Prepa"); ?>
</BODY>
</HTML>
```

- Sent to customer

```
<HTML>
<HEAD>
<TITLE> test </TITLE>
</HEAD>
<BODY>
2nd Y Prepa
</BODY>
</HTML>
```

Comments

- `//` for a line
- On several lines: starts with `/*` and ends with `*/`
- `#` as in the Unix Shell (on one line)
- Rmq : Avoid nesting comments

Instruction separator

- The semicolon: ;
- Except the last statement, the script terminator (eg: ?>) will mark the end of the last statement

Variables (1)

- PHP is not a strongly structured language:
- No well-defined declarative part.
- No declaration is necessary.
- A variable is always preceded by the symbol `$` : `$x` `$txt1` …

Variables (2)

- Supported types:
  - Scalars: booleans, integers, reals (nb floats), character strings, etc.
  - Compounds: arrays, objects
  - Specials: NULL
- The type is not declared, the engine decides according to the context

Variables (3)

- To create a variable, simply assign it a value.
- Subsequently, becomes accessible and will continue to exist until the end.
- Deletion of a variable ➔ Release memory space.
  - `unset ($ idf_var )`

boolean type

- `bool`
  - `TRUE, FALSE`
  - `0` or `0.0` or `""` or `"0"` or `NULL` ➔ `FALSE`
  - Empty array, empty object ➔ `FALSE`
  - Anything else even -1 ➔ `TRUE`

integer type

- `Integer / int`
  - Decimal base 10: `$x=141; $y=-141;`
  - Octal (radix 8): `$x=0215;`
  - Hexadecimal (base 16): `$x=0x8D;`
  - `FALSE` ➔ 0 and `TRUE` ➔ 1

real and string type

- Reals: (`float` / `double` / `real`)
  - `$pi = 3.14;`
- Character strings: (`string`)
  - `$ chn ="2 Lic ISIL"; $c='Welcome';`
- More details will be given in the character string management part.

Arrays

- Arrays in PHP are identified by a name which must respect the syntax of an identifier.
- Each entry in the array (index of an element) is called a key.
- Each value associated with an input (element) is called a value.
- Arrays in PHP are said to be associative (key, value)
- The key of an array can be integer (positive) or character string
  - `$tab['name']="Mohamed"; $tab[3]='Ali';`
  - `tab` is the name of the array, `name` and `3` are 2 keys
- `Mohamed` is the value associated with the key `name` and `Ali` is the value associated with the key `3`.

Creating a table (1)

- An array can be created directly by assignment:
  - `$tab[key1] = value1;`
  - `$tab[key2] = value2;`
- The key can be empty:
  - `$tab[]=val; // insert val into the`
    `// array at the 1st free key.`

Creating a table (2)

- An array can be created by the `array function` :
  - `$tab = array (key1=>val1, key2=>val2, …, clen => valn );`
  - `$tab = array (val1, val2, …, valn );`
  - This is equivalent to:
  - `$tab[0] = val1; $tab[1] = val2;…; tab[n-1] = valn ;`

Matrix

- You can also define arrays with 2 or more dimensions.
- Consider each element as another array.
  - `$tab['name'][1] = "Mohamed";`
  - `$tab['name'][2] = "Ali";`
- Note: At any time, elements of an array can be deleted: `unset ($tab[ key ]);`

The references

- Allows you to assign another name to a variable using the symbol `&` , For example:

```
$a=1; $b=&$a; // b and a : same thing
echo $a; // displays 1
echo $b; // also displays 1
$a = 2;
echo ($b); // will display 2
```

Dynamic variables

- In PHP, the name of a variable can in turn be variable.
- `$x="y"; $$x=1;` //it is the variable y which will have the value 1!!!
- `$z="tab"; ${$z}[1] =2;` //it is the 1st element of tab which will have the value 1
- `$vec[1] = "u"; $vec [2] = "v"; ${$vec[1]} = 3;` // it is the variable u which will have the value 3

The constants

- Environment Variables: Start with `$_`
- Example: `$_SERVER["SEVER_NAME"]`
- User-defined constants: `define (" idf_cst ", " val_cst ");`
- For example :
  - `define("PI", "3.14");`
  - `echo(PI);` //will display 3.14

Environment variables

- $_SERVER['AUTH_TYPE']
- $_SERVER['DOCUMENT_ROOT']
- $_SERVER['HTTP_ACCEPT_LANGUAGE']
- $_SERVER['HTTP_HOST']
- $_SERVER['HTTP_USER_AGENT']
- $_SERVER['PHP_SELF']
- $_SERVER['REMOTE_ADDR']
- $_SERVER['SERVER_ADDR']
- $_SERVER['REMOTE_PORT']
- $_SERVER['SERVER_PORT']
- $_SERVER['SCRIPT_FILENAME']
- $_SERVER['SCRIPT_NAME']
- $_SERVER['SERVER_PROTOCOL']
- $_SERVER['SERVER_SOFTWARE']

Forms data (1)

- To a form field:
  - which is called `field_name` ,
- They can be found in the tables:
  - `$_POST` and `$_GET`
  - This is according to the form submission method.

Form data (2)

- Each key in the array is the name of a field.
- Each value associated with a key is the value of the field as it was filled in, checked, chosen or selected by the user.
- It will be necessary to think of using isset for the verification of the sending of the data

PHP operators (1)

- Assignment =
  - `$a = 2;`
  - `$b = "this is an example";`
  - `$c = ($b = 4) = + 5; //4 → b and 9 → c`
  - `$tab[$i] = 3.14;`

PHP operators (2)

- Arithmetic:
  - + : addition
  - – : Subtraction
  - * : multiplication
  - / : Division (usually real)
  - % : Modulo

PHP operators (3)

- String operator:
  - . For concatenation
- Comparison :
  - == : equality
  - === : equality and same type
  - != : different
  - > : strict greater than >= : greater than or equal
  - < : strict less <= : less than or equal

PHP operators (4)

- Logics :
  - `And &&`
  - `Gold ||`
  - `xor`
  - `Not!`

PHP operators (5)

- Bits:
  - `&` : and
  - `|` : Or
  - `^` : or exclusive “ `xor` ”
  - `~` : no
  - `<<` : shift to the left by 1 bit (multiplication by 2)

Abbreviated Expressions

- `$x = $y = 2;` //assign 2 to x and y at the same time
- `$x++; ++$x;` // increment
- `$x--; --$x;` // decrement
- `$y=$x++;` //assignment of the value of x to y then increment of x
- `$y=--$x;` // decrement x then assign the value of x to y
- `$x .= "ENSIA";` //concatenate ENSIA to string in x

Converting variables (1)

- The PHP engine automatically performs the conversion as shown in the following example:
  - `$x = 1 + 2.5;` // of real type, x = 3.5
  - `$x = 1 + "2.5";` // of real type, x = 3.5
  - `$x = 1 + " abcd ";` // Error
  - `$x = 1 + "a2";` // Error
  - `$x = 1 + "10cm";` // of integer type, x = 11 with Warning
  - `$x = 0.5 + "10.5cm";` // of integer type, x = 11 with Warning

Converting variables (2)

- You can force the type of a variable in two different ways:
  - `settype($var, "type");`
  - `(type)$var;`
- Where type can be:
  - `int` , `integer` : for integers.
  - `bool` , `boolean` : for booleans.
  - `real`, `float` , `double` : for reals.
  - `string` : for character strings.
  - `array` : for tables.
  - `object` : for objects.

Converting variables (3)

- `$x = 3.6; settype($x," int "); //` x will have the value 3
- `$a = (int) (8/3); //` puts in a the value of the integer division of 8 by 3
- You can test the type of a variable using:
- `$t= gettype($var); //`returns in $t the type of $var //in the form of a char string
- You can also use the following Boolean functions:
- `is_long($x); is_double($x); is_string($x); is_array($x); is_object($x);`
- At any time, you can check the existence of a variable using the Boolean function: `isset($var);`

Character strings (quotes)

- In single quotes: `'` and `'`
  - Displays the text as it is.
  - `$x=2; echo 'This is an example $x';`
  - // Will give: This is an example $x
- `backslash (\)` character is used to precede the `'` and `\ characters` . To display them and avoid misinterpretations.

Character strings (quotes)

- Between double quotes (`"` and `"`)
  - Used to display the text as well as the content of variables and special characters:
  - `$x=2;`
  - `echo "This is an \"example\":\n $x";` //This will give:
  - // It's an example" :
  - // 2
- `\n` represents the line break.

Strings management (1)

- You can get the length of a string:
  - `$ln = strlen($ch );`
- We can compare two strings:
  - `strcmp ($ch1, $ch2);`
- You can access the ith character of a $ ch string :
  - `$c1 = $ch[$i-1];`
  - `$c2 = $ch[0];` // gives in c2 the 1st // character of the string

Strings management (2)

- Character strings can be processed using various functions including:
  - `str_replace ($ch1, $ch2, $ch3);`
    // replace in $ch3 any occurrence of $ch1 by $ch2
  - `addslashes($ch);` // add \ in front of any special character in $ch
  - `stripslashes($ch);` // remove the \ preceding the special characters

Regular expressions (1)

- Give each character and the number of occurrences.
  - `^` : indicates the beginning of a string.
  - `$` : indicates the end of a string.
  - Example: "^ start ", "end$", "^string$"
  - `[]` : defines the set to which a character belongs.
  - `[a]` or `a` : If it is exactly the character a.
  - `[abc]` : either the character `a` or `b` or `c` .
  - `[ac]` : from character `a` to character `c`
  - `{}` : allows you to define the number of occurrences of the previous character.
  - Example: `abc{2} = abcc , abc{2,} = abcc , abccc , abcccc … etc`

Regular expressions (2)

- `[ a|b|c ]` : Same as the previous 2 (or)
- `[0-9]` : Numeric character.
- `[az]` : Lowercase alphabetic character.
- `[AZ]` : Uppercase alphabetic character.
- `[a-zA-Z0-9]` : Alphanumeric character.
- . : Any character

Regular expressions (3)

- `?` : exactly one occurrence
- `*` : 0 or more occurrences
- `+` : at least one occurrence
- `{n}` : exactly `n` occurrences
- `{ n,m }` : at - `n` occurrence and at + `m` occurrences
- `{n,}` : at least `n` occurrences
- Example: a(bc)* = a, abc, abcbc … etc
- Example a date in the format: dd -mm- yyyy
  - `^[0-9]{2}\-[0-9]{2}\-[0-9]{4}$`
- Example identifier:
  - `^[a- zA -Z]?[a-zA-Z0-9]*$`

RegExp management (1)

- `preg_match("reg_exp", $ch);` : checks the match between the regular expression and the string `$ch`. Returns `TRUE` or `FALSE` .
- For example :
  - `preg_match("#^[A-Za-z0-9_.-]{4,20}$#", $user);`
  - `#` is just a separator, we could have put `\` or `!` or `§`
  - `preg_match("#^[a-z0-9_.-]{4,20}$#i", $user);`
  - We added the i to ignore case

RegExp management (2)

- There are several regular expression processing functions:
  - `preg_replace("reg_exp", $ch1, $ch2) ;` : replaces in `$ch2` any occurrence of a string matching the regular expression with the string defined in `$ch1` .
  - `$tab= preg_split("reg_exp", $ch);` : splits `$ch` into multiple substrings relative to the characters defined by the regular expression. Each substring is saved in the `$tab array` .

Array management

- PHP has several functions for managing and processing arrays:
- `sort($tab)` : sorts an array against values in order.
- `rsort ($tab)` : sorts an array by values in reverse order.
- `ksort ($tab)` : sorts an array against the keys in order.
- `krsort ($tab)` : sorts an array against the keys in reverse order.
- `reset($tab)` : resets the pointer to the 1st element of the array.
- `end($tab)` : positions the pointer on the last element of the array
- `$nb= sizeof ($tab);` : returns the number of elements of the array `$tab`

Conditional statement `if`

- `if ( logical_expression ) statement_block`
- Example :

```
if ($x>0)  $x--;
 if ($x<=0)  {
  $x++;
  echo $x;
 }
```

Instruction conditionnelle `if else`

```
if (logic_exp) instructions_bloc_1;
else instructions_bloc_2;
```

- Example :

```
if ($x>0) $x--;
else   $x++;
if ($x<=0) {$x++; echo $x; }
else   {$x--; echo $x; }
```

Conditional statement `if elseif`

- To avoid nested `if` we have the following statement:

```
if (cond_1) { instruction_block_1;}
 elseif (cond_2) {block_instructions_2;}
…
 elseif ( cond_n ) { instruction_block_n ;}
 else {block_instructions_n+1;}
```

Abbreviated conditional statement

- `$var = Expression? Val1: Val2;`
- Equivalent to:

```
if (Expression)
{$var = Val1;}
else
{$var = Val2;}
```

switch statement

```
switch($var) {
case val_1: bloc_instr_1; break;
...
case val_n : bloc_instr_n ; break;
default: bloc_instr_n+1;
}
```

while loop

- As long as the condition is true it will execute the instructions.

```
while (condition) {
inst_1;
inst_2;
…
 inst_n ;
}
```

- With a single instruction, we can remove the braces

Do- while loop

- Executes the statements then evaluates the condition. If it is checked, it continues otherwise it stops.

```
do {
inst_1;
inst_2;
…
 inst_n ;
} while (condition);
```

- With a single instruction, we can remove the braces

`for` loop

```
for (expr1; expr2; expr3) {
inst1;
inst2;
…
instn ;}
```

- Execute `expr1` before starting the loop, test `expr3` before each iteration, then execute `expr2` at the end of each iteration.
- With a single instruction, the braces can be removed.

foreach loop (1)

- To browse tables:

```
 foreach($idf_tab as $val) {
Inst_1;
Inst_2;
…
  Inst_n ;}
```

- Iterates through all values in the `$idf_tab array` .

`foreach` loop (2)

- Start by pointing to the first element.
- On each iteration saves the value of the current array element to `$val` .
- At the end of each iteration points to the next element in the array
- Ends after going through all the elements in the array.
- With a single instruction, the braces can be removed.

foreach loop (3)

- There is another form of `foreach` :

```
 foreach ($ idf_tab as $ key =>$val) {
Inst_1;
Inst_2;
…
  Inst_n ;}
```

- Same as the first form. Except that it additionally saves the key of the current item in `$ key` .

Loops – continued

- We can use `break` and `continue`
  - `break` : exits the loop without checking the stop condition
  - `continue` : allows you to abandon the current iteration and go directly to the next iteration

Definition of a function (1)

- Definition of the function:

```
 function idf_funct ($arg1,$arg2,…,$ argn ){
inst_1;
inst_2;
…
  inst_n ;
return $var; }
```

Definition of a function (2)

- The `return statement` allows the function to return a value.
- `return` can be inserted anywhere in the function.
- We can have a function without arguments: `function idf_fct () { instr …}`
- We can have a function that does not return values (procedure)

Return multiple values

- We use an array as a variable:

```
return $tab;
```

  - `$tab` must be of type array.
- This array can be created when using `return` :

```
return array ($key_1=>$val_1, …, $ key_n =>$ val_n );
return array ($val_1, …, $ val_n );
```

Default settings

- An argument can have a default value when defining the function:

```
function f ($x, $y, $z=1) { statements…}
```

- The arguments defined by default must be last in the list of arguments.

Call of a function

- The call to a function must always be made after its definition:

```
$ idf_var = idf_function (arg1, …, argn );
$ idf_function (arg1,…, argn );
```

- During the call, the arguments are passed in the following ways:
  - By values
  - By references: permanently or not permanently

Passing arguments by values

- Passing by values is an ordinary pass:
  - There is nothing special to add during the definition.
  - When calling, we put values or variables as arguments
  - Variables are input.

Passing arguments by references

- Passing by permanent references is done during the definition of the function.
  - When defining, we precede the argument variable of & .
  - There is nothing special to add when calling.

Passing arguments - example

- `function f1($x) { $x++;}`
- `function f2(&$y) { $y++;} // def . 1 passage always by references: permanent.`
- `//////pass by value`
- `$a=1; f1($a); echo $a; // will output 1. a as input`
- `//////passing by references not permanent`
- `$a=1; f2($a); echo $a; //will display 2. a in input/output`

Global/local variables

- Global variable: visible and recognized in all PHP code ➔ `aggregate $var; $GLOBALS["var"];`
- Static variable: specific to the function, but it is recognized throughout the execution of the PHP code. Keep value on next call ➔ `static $var;`

Local var. – example

```
$a = 1;
function f1()
{
 echo $a;
}
f1();
```

- Displays nothing because it is the variable a of the function f1 which does not contain any value and not a of the main code.

Global var. – example 1

```
$a = 1;
function f2() {
overall $a, $b;
$b = $a++;
}
f2();
echo $a;
echo $b;
// Will print 2 for a and 1 for b
```

Global var. – example 2

```
$a = 1;
function f3() {
$GLOBALS["b"] = $GLOBALS["a"]++;
}
f3();
echo $a;
echo $b;
// Will display 2 for a and 1 for b
```

Static var. - example

```
$a = 1;
function f4() {
static $a=2;
$a++;
echo $a;
}
f4(); // Will display 3
echo $a; // will display 1
f4(); // Will display 4
```

Some predefined functions (1)

- Predefined functions are not case sensitive ➔ can be written in upper or lower case.
- We will focus mainly on the mathematical functions that are the most used
- `$y=abs($x);`

Returns the absolute value of `$x`

`$x` and `$y` are numbers

Some predefined functions (2)

- `$y= ceil ($x);`
  - Returns the integer immediately greater than `$x` (round up)
  - `$x` and `$y` are reals
- `$y= floor ($x);`
  - Returns the integer part of `$x` (rounded up)
  - `$x` and `$y` are reals
- `$y=round($x, $N);`
  - Round `$x` to `$N` decimal places.
  - `$x` and `$y` are real numbers and `$N is` an integer.

Some predefined functions (3)

- `$z= fmod ($ x,$y );`
  - Returns the (real) remainder of the division of `$x/$y`
  - `$x, $y` and `$z` are reals.
- `$y= sqrt ($x);`
  - Returns the square root of `$x`
  - `$x` and `$y` are reals
- `$y= pow ($ x,$n );`
  - Returns $x^{\$n}$
  - `$n, $x` and `$` y are numbers

Predefined functions

- PHP has a multitude of functions.
- These functions are stored in extension modules.
- It is possible to check the modules installed on the server:
- `$tab= get_loaded_extensions ();`
  - Returns an array, each entry of which contains the name of an available module.
- It is also possible to list the functions of a given module:
- `$tab= get_extension_funcs ($module-name);`
  - Returns an array, each entry of which is a character string containing the name of a module function.

Date management (1)

- The original date is 01/01/1970 at 00:00'00".
- A counter of the number of seconds between this date and the current date.
- This number of elapsed seconds is called “ `timestamp` ” or “ `Unix instant` ” .
- There are several PHP functions for processing dates:
- `$t=time();`
  - Returns the current timestamp (`$t` is integer).

Date management (2)

`$d= getdate ($t);`

- Returns the date (`$d`) as an array. `$t` is a timestamp (integer).

- `$d[' wday ']` : weekday from 0 to 6 (0=Sun.)
- `$d[' weekday ']` : weekday in English.
- `$d[' mday ']` : day of the month from 1 to 31.
- `$d['mon']` : the month from 1 to 12 (1=jan.).
- `$d[' month ']` : the month in English.
- `$d[' year ']` : the year (integer on 4 positions).
- `$d[' hours ']` : hour from 0 to 23.
- `$d['minutes']` : minutes from 0 to 59.
- `$d['seconds']` : seconds from 0 to 59.
- `$d[' yday ']` : the day of the year from 1 to 366.

Date management (3)

- `$b= checkdate ($month, $day, $year);`
  - Returns a boolean (`$b`).
  - `$month` , `$day` and `$year` are 3 integers representing a date.
  - Used to check whether the date given in parameters is valid.
  - Returns `TRUE` , if it is actually a date.
  - `checkdate (2, 30, 2022)` will return `FALSE` because February 30, 2022 is not a date.

File management (1)

- Managing files in PHP is done almost the same way as in C:
- File existence test
- `$b= file_exists ($ file_name );`
  - Returns a boolean, `TRUE` if the file exists.
- Creating a file
- `touch ($ file_name , $t);`
  - Allows you to create a file whose name is `$ file_name` and the last modified date is the timestamp `$ t` .
  - Pay attention to the read permission of the file.

File management (2)

- Verifications:
- `$b= is_file ($ file_name );`
  - Returns a boolean, `TRUE` if it's a file.
- `$b= is_readable ($ file_name );`
  - Returns `FALSE` if it is read-protected.
- `$b= is_writable ($ file_name );`
  - Returns `FALSE` if it is write-protected.
- `$b= is_uploaded_file ($ file_name );`
  - Returns `TRUE` if it was sent by `form` .
- `$type= filetype ($ file_name );`
  - Returns the file type (char string)

File management (3)

- Opening a file ➔ Necessary before any operation.
- `$f= fopen ($ file_name , $mode, $ path );`
  - Opens a file in the specified mode.
  - It returns a resource type variable.
  - The resource type is a special type.
  - It will allow to perform all possible operations on the file by referring only to the resource variable and without quoting the file name.

File management (4)

- `$mode` is a string as follows:
  - `r: read` , indicates an opening for reading.
  - `w: write` , indicates an opening for writing. Overwrites the content.
  - `a: append` , indicates an opening for writing with the content appended to the end of the file.
- when mode is followed by [b] ➔ file is treated as binary.
- modes a and w allow the creation of the file if it does not exist.

File management (5)

- The `$ path parameter` is optional, if it is 1 or TRUE, the search for the file will extend to sub-directories.
- `fopen` also allows the opening of files located on other sites (another server). Just give the full URL as the filename.

File management (6)

- Locking a file ➔ This is important on the Web since there is a risk of multiple accesses to the file:
- `$b= flock ($f, $n);`
  - Returns a boolean. TRUE if the lock operation was successful and FALSE otherwise.
  - `$f` is the resource variable used when opening.
  - `$n` is an integer that takes 3 values: 1 to lock write, 2 to lock read and write, and 3 to unlock.

File management (7)

- Reading ➔ several functions:
- `$ ch = fgets ($f, $ num_bytes );`
  - Reads the file (from the beginning) and returns a character string `$ ch` of size `$ nb_bytes` bytes at most.
  - `$f` resource variable.
  - Reading stops when it encounters a newline or the end of file and returns 0 on failure.

File management (8)

- Reading (continued):
- `$ ch = fread ($f, $ num_bytes );`
  - Reads the file and returns a character string `$ ch` of size equal to `$ nb_bytes` bytes.
  - `$f` resource variable.
  - Reading stops when the end of file is encountered.
- `$c= fgetc ($f);`
  - Reads the file and returns a character (`$` c).
  - `$f` resource variable.

File management (9)

- Writing ➔ several functions:
- `fputs ($f, $ ch , $n);`
  - Writes the contents of the string `$ch to the file .`
  - `$n` parameter is optional.
  - If specified, only the first `$n` characters of `$ ch` will be written.
  - `$f` resource variable.

File management (10)

- Writing (continued):
- `fwrite ($f, $ ch , $n);`
  - Writes the contents of the string `$ch to the file .`
  - `$n` parameter is optional.
  - If specified, only the first `$n` characters of `$ ch` will be written.
  - `$f` resource variable.

File management (11)

- Checking for end of file:
- `$b= feof ($f);`
  - Returns a boolean (`$b`) if the end of file was reached while reading.
  - `$f` resource variable.
- File size:
- `$size= filesize ($f);`
  - Returns size in bytes.
  - `$f` file name .

File management (12)

- Closing a file ➔ Necessary before exiting the program (script):
- `$b= fclose ($f);`
  - Returns a boolean (`$ b`)
  - TRUE if the operation was successful.
  - `$f` resource variable (created when opening with `fopen`).

Files with form (1)

- Upload a file to the server from the client using a form.
- We use the `<input> field` of `type="file"` .
- The form must be sent with the post method.
- You must add the `enctype =" multipart / form -data "` attribute to the `<form> tag` .

Files with form (2)

- After sending the file through the form:
- The file will be saved on the server in a buffer directory.
- This directory is defined in `php.ini` using the `upload_tmp_dir directive`.
- It is saved under a different randomly generated name.
- If no script handles it immediately, it will be lost.

Files with form (3)

- To find the file, we have the associative array `$_FILES` .
- It is a 2 dimensional array.
- The 1st key: name assigned to the "file" form field.
- The 2nd key indicates the information needed to process the transferred file.
- If for example the field is called "f1", we will have:

Files with form (4)

- (Original) name of the file on the client: `$_FILES["f1"][" name "]` .
- File MIME type: `$_FILES["f1"]["type"]` .
- Size in bytes: `$_FILES["f1"]["size"]` .
- Temporary name of the file on the server: `$_FILES["f1"][" tmp_name "]` .
- The error code (4 values defined by constants): `$_FILES["f1"][" error "]` .

Files with form (5)

- To save the file and rename it:
- `move_uploaded_file ($_FILES["f1"][" tmp_name "], final_file_name ) .`
- Returns a boolean: TRUE if the operation was successful, FALSE otherwise.

Tips – multiple values

- For checkboxes, multiple-selection list boxes…
- Multiple user-selected values and a server-level variable: `$_POST[" field_name "]` or `$_GET[" field_name "]` .
- Solution: In the form, we add the characters `[] to the name of the field` . For example: `name ="option[]"` .
- In the script, we retrieve the different values in the form of an array:
- For example `$_POST[" field_name "][0]` or `$_GET[" field_name "][0]` is the 1st value selected or checked.

Cookies – overview

- Small files that can be inserted into the client's machine.
- Does not exceed 4 KB.
- An X server cannot write more than 20 cookies to a Y client.
- Can be deactivated by the customer!
- May have several interests: saving data, ensuring that it is the same customer, advertising, etc.

Cookies – setcookies (1)

- In PHP, we have the function (create or delete):
- `setcookie ( c_name , c_value , end_date , path, domain, security );`
- Only the 1st argument which is mandatory.
- To create the cookie, you must give at least a name and a value.
- To destroy the cookie, you just have to give the name.

Cookies – setcookies (2)

```
setcookie ( c_name , c_value , end_date , path, domain,
security );
```

- `end_date` : Defines the date (timestamp) from which the cookie will no longer be usable. For example time()+86400 to give it a lifetime of 24H.
- `path` : path to the folders whose scripts are allowed to use this cookie.

Cookies – setcookies (3)

- `domain` : domain name. (if it is another site)
- `security` : TRUE if the cookie is transmitted via a secure connection.
- `setcookie (…)` : Returns a boolean.

Cookies – setcookies (4)

- To pass multiple values in the same cookie,
- For example :
- `setcookie (" student [name]", "BENOMAR", time()+3600);`
- `setcookie (" student [ firstname ]", "ALI", time()+3600);`
- `setcookie (" student [ faculty ]", "FEI", time()+3600);`

Cookies – reading

- To read cookies, access the array: `$_COOKIE[name]`
- If several values have been transmitted in the same cookie
- We can use the `foreach loop` on the array `$_COOKIE[name]`
- Or access directly such as:
- `$_COOKIE[" student "]["name"]`

Sessions (1)

- HTTP: stateless transmission protocol.
- Nothing allows him to know if 2 requests come from the same client computer or not.
- Keeping information from one page to another?
- Sessions mechanism introduced since PHP4
- Sessions with cookies or Sessions without cookies

Sessions (2)

- Login.
- Allocation of a unique identifier: transmitted from one page to another (by cookie or added to the URL)
- Definition of session-related variables stored in a folder on the server.
- Reading/writing session variables.
- Closing session / Destroying session variables.

Sessions (3)

- Begin each page with:

```
session_start ();
```

- Defining session variables:

```
$_SESSION[' var_name '] = value;
```

- Access to the session variable by the array: `$_SESSION`
- To destroy all session variables:

```
session_unset ();
session_destroy ();
```