Introduzione PHP

Introduzione

- PHP è acronimo di *Hypertext PreProcessor*.
- PHP è un linguaggio tuttora utilizzato, anche se negli ultimi anni ci si è messi verso linguaggi alternativi.
- PHP è un linguaggio di scripting orientato al lato server, gli script sono eseguiti lato server e non abbiamo accesso al codice.
- PHP è un software open source: può essere scaricato e usato liberamente.
- È compatibile con quasi tutti i server web (Apache, IIS, …. Noi utilizzeremo Apache)
- Per utilizzare PHP dobbiamo installare un web server, precisamente Apache, attraverso lo zip fornito dal docente (solo per chi ha Windows, chi ha Mac si arrangia)

File PHP

- Un file PHP può contenere testo, tag HTML e ovviamente codice PHP. La struttura può essere decisa dinamicamente attraverso l'apertura e la chiusura di tag PHP (ciò che sta dentro questi tag è codice PHP, al di fuori tutto viene stampato come se fosse codice HTML).
- I file PHP sono restituiti dal browser come pagine HTML.
- Solitamente i file PHP presentano una delle seguenti estensioni: `.php`, `.php3` `.phtml`.
- In un file con estensione `.html` il codice PHP non sarà eseguito.

PHP e Databases

- PHP offre un'integrazione con diversi tipi di database (primo fra tutti MySQL, che vedremo)
- La combinazione PHP-MySQL è *cross-platform*: possiamo sviluppare i nostri servizi web su piattaforme diverse.

Inclusione del PHP in una pagina HTML

- Possiamo includere codice PHP in uno dei seguenti modi
  - `<?php echo 'contenuto'; ?>`
  - `<? ISTRUZIONE ?>`, oppure `<?= expression ?>` (quest'ultima equivalente ad `echo`)
- Il primo metodo è quello principale, il secondo potrebbe essere disponibile solo se attivato attraverso il file di configurazione php.ini.
- Nella realizzazione del progetto è caldamente sconsigliato alterare la configurazione di php.ini: se indispensabile segnalarlo.

Istruzioni in PHP

- Ogni istruzione in PHP deve concludersi con un punto e virgola, contrariamente al Javascript dove il punto e virgola è facoltativo (anche se consigliato).
- La chiusura di un blocco di codice PHP implica automaticamente un punto e virgola.

Commenti

- I commenti possono essere espressi nei seguenti modi:
  - // per commenti su singola riga
  - /* per commenti larghi */
  - # per commenti su singola riga (*as in Unix shell*)

Variabili in PHP

- Tutte le variabili in PHP sono introdotte con il simbolo dollaro $

```
$var_name = value;
```

- Le variabili in PHP non devono essere dichiarate: il PHP converte la variabile nel corretto *data type* in base al suo valore. Non si ha la tipizzazione forte del C++, come in Javascript.
- PHP è case-sensitive relativamente alle variabili: il primo carattere deve iniziare con una lettera o un underscore _. Sono accettati solo caratteri alfanumerici e underscore.
- Si raccomandano nomi intuitivi che permettano di riconoscere in modo agile il contenuto della variabile.

PHP Types

- PHP supporta otto tipi primitivi. Precisamente abbiamo
  - Quattro tipi scalari:
    - **Boolean**
      - Può avere come valore TRUE o FALSE.
      - Le keyword sono case-insensitive
    - **Integer**
      - Può essere espresso come decimale, esadecimale od ottale.
      - Il numero può essere preceduto da un segno (+ o -)
      - La dimensione di un intero può essere determinata con le costanti PHP_IN_SIZE.
      - Il valore massimo di un intero può essere determinato con la PHP_INT_MAX.
    - **Float**
      - Stesse notazioni di Javascript: notazione esponenziale o notazione in virgola mobile.
    - **String**
      - Sequenza di caratteri.
      - Le stringhe sono individuate da doppi apici o da singoli apici.
      - Con l'uso dei doppi apici i nomi delle variabili sono espansi: questo significa che ponendo all'interno il nome di una variabile non stamperò questo ma il suo contenuto!
      - Se non possiamo fare quanto detto prima possiamo ricorrere alla cosiddetta `complex curly syntax` (esempi alla diapositiva 17)
      - Possibile l'uso della *Heredoc syntax*. Possiamo scrivere la variabile su più riga attraverso la seguente sintassi

```
<?php
$str = <<<EOD
riga1
riga2
riga3
EOD;
echo $str;
?>
```

      - Il punto consiste nell'operatore di concatenazione

```
$var = $var1.$var2;
```

        si ha come valore di $var la concatenazione di $var1 e $var2
      - La concatenazione con assegnamento si fa con l'operatore .=

```
$testo = "Hello";
$testo .= " world";
echo $testo; // "Hello world"
```

      - La stringa può essere immaginata come un array: possiamo lavorare sui singoli caratteri, estrarli e/o modificarli, e ottenere il numero di caratteri attraverso la funzione `strlen($str)`
      - Con la funzione `strpos("testo"; $variabile)` possiamo individuare eventuali corrispondenze all'interno di una variabile. Se si trovano corrispondenze si restituisce l'indice della posizione, in caso contrario si restituisce FALSE.
      - Presenti altre funzioni a pagina 22. In particolare ci interessano le seguenti funzioni:
        - `echo`
          consente di stampare in uscita gli argomenti. In caso di un solo argomento

non serve utilizzare le parentesi, altrimenti sono obbligatorie.

- `explode("delimiter", $variable)`
  genera un array che consiste in parti della stringa $variable: questa è stata divisa attraverso il delimitatore indicato.

- `implode("delimiter", $array)`
  genera una stringa a partire da un array: si concatena il contenuto dei vari elementi dell'array separandoli attraverso il delimitatore indicato (processo inverso della explode)

- `number_format($number, $decimals, $dec_point, $thousands_sep)`
  permette di generare una stringa contenente all'interno un numero reale. Il numero di parametri può variare:
  - Un parametro: il numero viene formattato senza decimale, ma con una virgola che separa terne di cifre.
  - Due parametri: il numero sarà formattato con decimali (separati dalla parte intera col punto). Le terne di cifre della parte intera sono trattate come già detto.
  - Quattro parametri: il numero sarà formattato con decimali. La parte intera e quella decimale sono separate dall'elemento indicato negli argomenti, stessa cosa per le terne di cifre nella parte intera.

- `parse_str($str, $output)`
  fa l'analisi della stringa come se fosse passata via URL e inizializza delle variabili. Il numero di parametri può variare:
  - con un parametro abbiamo l'inizializzazione di parametri con i valori effettivamente indicati nell'URL
  - con due parametri definiamo un array contenente tutte le variabili definite dall'URL.

  - **Esempio di URL**: "first=value&arr=[]=foo+bar&arr[]=baz"

- `strncmp($str1, $str2, $len)`
  funzione per il confronto di variabili. Uguale alla funzione in C++: unica differenza è la possibilità di indicare un limite al numero di caratteri da comparare nelle due stringhe. Come al solito:
  - Se il risultato è negativo $str1 è minore rispetto ad $str2
    Se il risultato è positivo $str1 è maggiore rispetto ad $str2
  - Se il risultato è 0 le due stringhe sono uguali

- Due tipi speciali:
  - **NULL**
    - Rappresenta una variabile nulla, cioè senza valore.
    - Possiamo porre questo valore assegnando la costante NULL o mediante la funzione unset. Generalmente una variabile non inizializzata ha come valore NULL.
    - La funzione is_null() permette di valutare se un valore è nullo.

  - **Resource**
    - Variabile speciale che permette di comunicare con una risorsa esterna. Un esempio si ha relativamente ai database.

- Due tipi composti:
  - **Array**
    - Un array PHP è una mappa ordinata: il valore si recupera attraverso una chiave (che non è detto sia per forza un intero)

- Questa cosa è ottima per usi differenti: array, list (vector), hash table, dictionary, collection, stack, queue.
- Gli array possono essere multidimensionali: elementi di array possono essere a loro volta array.
- Una variabile può essere inizializzata come array attraverso il costrutto array(). Poniamo in ingresso il contenuto dell'array: abbiamo una lista di elementi, separati da virgola, nella seguente forma
  `key => value`
  possiamo limitarci a inserire soltanto una lista di value: a quel punto le chiavi identificative consisteranno in interi da 0 in su (si ha un *array numerico*, come in un qualunque array in C++)

  ```
  $first_example = array("nome" => "Gabriele", "cognome" => "Frassi);
  $second_example = array(10, 8, 3, 5, 6);
  echo $first_example['nome']; // Gabriele
  echo $second_example[1]; //8
  ```

  i due array possono essere modificati in modo estremamente flessibile
  ```
  $first_example['nome'] = "Giuseppe";
  $second_example[] = 52; // Ho aggiunto un nuovo elemento in fondo all'array
  unset($second_example[0]); // ho rimosso il primo elemento, gli indici non scalano
  unset($second_example); // Ho eliminato l'intero array
  ```

- `array_fill($start_index, $num; $value)`
  funzione che permette di riempire l'array con un certo numero di elementi aventi tutti lo stesso valore, a partire da una certa posizione.

- `array_combine($keys, $values)`
  crea un array dati in ingresso due array paralleli: dal primo array si prendono le chiavi, dal secondo i valori.

- `array_keys($array)`
  restituisce un array numerico contenente le chiavi dell'array passato per parametro

- `array_values($array)`
  restituisce un array numerico contenente i valori dell'array passato per parametro

- `array_merge($array1, $array2, …)`
  permette di unire due o più array. Si tenga conto delle seguenti situazioni di conflitto:
  - se alcuni elementi dell'array hanno la stessa chiave il valore più recente sovrascriverà quello più vecchio;
  - se ci troviamo in array con chiavi numeriche i valori più recenti non sovrascriveranno quelli originali, ma saranno aggiunti.

  In un array numerico le chiavi saranno reimpostate in modo incrementale partendo dal numero zero.

- `gettype($variable)`
  restituisce una stringa contenente il tipo della variabile

- `is_int($an_int)`
  restituisce un booleano che indica se la variabile è un intero o no.

- `is_string($a_bool)`
  restituisce un booleano che indica se la variabile è una stringa o no.

- `settype($variable, "type")`
  funzione che converte in modo forzato la variabile $variable nel tipo passato come secondo parametro. Per esempio:

```
$foo = "5bar";
$bar = true;

settype($foo, "integer"); // Ottengo 5
settype($bar, "string"); // Ottengo "1"
```

- `Funzioni di ordinamento`: pongo qua di seguito una tabella contenente le principali funzioni per l'ordinamento. Per maggiori informazioni visitate la pagina dedicata su *`php.net`*. Scegliete quale funzione adottare in base alle vostre esigenze: ordinamento in base alle chiavi o ai valori, mantenimento delle chiavi o meno dopo l'ordinamento, criterio di ordinamento.

Nome funzione	Sorts by	Mantenimento delle chiavi	Order of sort
asort()	Valore	yes	Dal più piccolo al più grande.
arsort()	Valore	yes	Dal più grande al più piccolo.
krsort()	Chiave	yes	Dal più grande al più piccolo.
ksort()	Chiave	yes	Dal più piccolo al più grande
rsort()	Valore	no	Dal più grande al più piccolo.
shuffle()	Valore	no	Ordinamento casuale
sort()	Valore	no	Dal più piccolo al più grande
uasort()	Valore	yes	Definito dall'utente con una funzione callback.
uksort()	Chiave	yes	Definito dall'utente con una funzione callback.
usort()	Valore	no	Definito dall'utente con una funzione callback.

  - **Object**: istanze di classi (come in C++).

Variable scope

- Contrariamente al Javascript una variabile usata all'interno di una funzione è di default limitata allo scope locale della funzione. Questo significa che con un codice del genere

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

  la funzione test non può leggere il valore di $a esterno alla funzione: posso leggere solo una variabile $a locale alla funzione.

- **Rimedio**: se poniamo all'interno della funzione la seguente istruzione
  `global $a;`
  stabiliamo con la keyword global di andare a leggere il valore della variabile $a esterna alla funzione.
- Si consiglia di usare con responsabilità quest'ultima istruzione.

- Possiamo ottenere lo stesso risultato usando un array predefinito di PHP: `$GLOBALS` (tutto maiuscolo). Consiste in un array associativo che permette di recuperare una variabile globale. Se io pongo
  `$GLOBALS['a']`
  posso recuperare il valore della variabile $a, che è 1.

- **Variabili statiche**: possiamo introdurre variabili statiche attraverso la keyword static
  `static $a = 0;`
  variabile limitata alla funzione ma con tempo di vita coincidente a quello del codice che stiamo eseguendo (come in C++)

Variabili ed array predefiniti (Superglobals)

- I *superglobals* consistono in variabili built-in sempre disponibili in tutti gli scopes.
- Cosa scritte come spiegate da Marcelloni. Non tutte le cose sono vitali.
- Abbiamo:
  - `$GLOBALS`, tutte le variabili disponibili nello scope globale
  - `$_SERVER`, informazioni relative al server
  - `$_GET`, array associativo di variabili passate attraverso l'URL (IMPORTANTISSIMO!)
  - `$_POST`, array associativo di variabili passate attraverso form (IMPORTANTISSIMO!)
  - `$_FILES`, array associativo che contiene i files caricati attraverso il metodo POST
  - `$_REQUEST`, array associativo che contiene quanto presente nei tre array precedenti
  - `$_SESSION`, array associativo che contiene variabili di sessione (importanti, soprattutto quando gestiamo sistemi di login). Il loro tempo di vita è limitato.
  - `$_ENV`, array associativo di variabili passate allo script corrente dallo SHELL (Windows, Unix)
  - `$_COOKIE`, array associativo di variabili passate al codice attraverso gli HTTP Cookies
  - `$php_errormsg`, variabile contenente l'ultimo errore generato dal PHP Disponibile solo se viene attivata l'opzione *track_errors* nel php.ini

  - `$HTTP_RAW_POST_DATA`, dati grezzi relativi al metodo post
  - `$http_response_header`, intestazione del pacchetto http
  - `$argc`, numero di argomenti passati allo script quando eseguito da una linea di comando
  - `$arv`, numero di argomenti passati allo script

Riferimenti

- Di default tutte le variabili sono assegnate per valore
- Possibile di assegnare come valori dei riferimenti. I concetti sono simili a quelli del C++.

```
$foo = 'Bob';
$bar = &$foo;
$bar = "Ciao";
echo $bar;
echo $foo;
```

  Le variabili risulteranno uguali: modificando bar modifichiamo anche foo (tutto previsto, come in C++)
- **Operazioni possibili**:
  - Assegnamento per riferimento
  - Passaggio alla funzione per riferimento
  - Valore restituito per riferimento
- L'unico modo per separare una variabile dal suo riferimento è utilizzare la funzione `unset()`, già introdotta.

Costanti

- Le costanti hanno nomi che iniziano con una lettera o un underscore, seguita da un qualunque numero di lettere, numeri e/o underscores.
- È possibile introdurre nuove costanti attraverso la funzione `define()`. Vediamo degli esempi

```
define("FOO", "something");
define("FOO2", "something else");
define("FOO3", "something more");
```

- PHP offre un certo numero di costanti predefinite (tutte precedute e seguite da due underscore):
  - __LINE__, numero di linea corrente del file
  - __FILE__, percorso completo e nome del file che stiamo eseguendo
  - __DIR__, percorso del file
  - __FUNCTION__, nome della funzione eseguita (case-sensitive)
  - __CLASS__, nome della classe (case-sensitive)
  - __METHOD__ , nome del metodo della classe (case-sensitive)

Operatori

- Gli operatori in PHP sono sostanzialmente gli stessi visti in C++.

- **Operatori di confronto**:

Example	Name	Result
$a = = $b	Equal	TRUE if $a is equal to $b.
$a = = = $b	Identical	TRUE if $a is equal to $b, and they are of the same type. (introduced in PHP 4)
$a != $b	Not equal	TRUE if $a is not equal to $b.
$a <> $b	Not equal	TRUE if $a is not equal to $b.
$a != = $b	Not identical	TRUE if $a is not equal to $b, or they are not of the same type. (introduced in PHP 4)

- **Operatori logici**:

Example	Name	Result
$a and $b	And	TRUE if both $a and $b are TRUE.
$a or $b	Or	TRUE if either $a or $b is TRUE.
$a xor $b	Xor	TRUE if either $a or $b is TRUE, but not both.
! $a	Not	TRUE if $a is not TRUE.
$a && $b	And	TRUE if both $a and $b are TRUE.
$a || $b	Or	TRUE if either $a or $b is TRUE.

- **Operatori array:** (novità rispetto a Javascript e C++)

Example	Name	Result
$a + $b	Union	Union of $a and $b.
$a == $b	Equality	TRUE if $a and $b have the same key/value pairs.
$a === $b	Identity	TRUE if $a and $b have the same key/value pairs in the same order and of the same types.
$a != $b	Inequality	TRUE if $a is not equal to $b.
$a <> $b	Inequality	TRUE if $a is not equal to $b.
$a !== $b	Non-identity	TRUE if $a is not identical to $b.

  - L'unione permette di unire due array. In caso di collisione di elementi aventi la stessa chiave si utilizzeranno i valori del primo array scartando quelli del secondo.
  - L'uguaglianza mi permette di stabilire se hanno in comune tutte le coppie chiave-valore
  - L'identità oltre a svolgere i controlli dell'uguaglianza verifica se chiavi e valori sono degli stessi tipi
  - Ovviamente abbiamo anche la disuguaglianza e la non uguaglianza.

  - Attenzione al seguente esempio: gli array non sembrano uguali ma in realtà lo sono! Questo perché le chiavi degli elementi del secondo array coincidono con le chiavi del primo array.

```
$a = array("apple", "banana");
$b = array(1 => "banana", "0" => "apple");
var_dump($a == $b); // bool(true)
var_dump($a === $b); // bool(false)
```

    Conclusione: l'ordine degli elementi non è decisivo nel determinare l'uguaglianza di due array.

- **Differenze**:
  - Prima di tutto gli operatori di uguaglianza e disuguaglianza esatta già visti in Javascript
  - Il diverso può essere fatto anche come in MySQL (!= o <>)

  - **Operatore di controllo degli errori**: porre la chiocciola @ prima di un'espressione in PHP permette di ignorare messaggi di errori generati da quella espressione. Tuttavia se la feature `track_errors` è attivata nel `php.ini` un qualunque errore generato dall'espressione sarà salvato nella variabile

```
$php_errormsg.
```

- **PHP supporta i backticks** come **operatore di esecuzione** di comandi UNIX. PHP cercherà di eseguirli come comandi shell. L'output sarà restituito attraverso la variabile usata. Attenzione: se la *safe mode* è abilitata o `shell_exec()` disattivata i *backticks* non funzioneranno.

- Negli operatori logici abbiamo due versioni diverse sia dell'operatore AND che dell'operatore OR. Esse operano seguendo differenze precedenti: bisogna tenerne conto se vogliamo salvare in una variabile il risultato di un'espressione logica

```
$e = false || true; // Acts like: ($e = (false || true))
$f = false or true; // Acts like: (($f = false) or true)
```

Operatore *istanceof*

- L'operatore `istanceof` restituisce un risultato booleano che indica se l'oggetto è istanza di una classe.

- **Esempio**:

```
class MyClass {}
class NotMyClass {}
$a = new MyClass;
var_dump($a instanceof MyClass); // bool(true)
var_dump($a instanceof NotMyClass); // bool(false)
```

PHP Statements

- Gli statements del PHP sono praticamente uguali a quelli del C++.
- Essi sono:
  - *if*
  - *switch*
  - *while*
  - *do-while*
  - *for*
  - *foreach*
  - *break*
  - *continue*
  - *include*
  - *require*
  - *include_once*
  - *require_once*

If-statements

- Sintassi praticamente uguale
- Esempio di codice:

```
if (condition)
    code to be executed if condition is true;
elseif (condition)
    code to be executed if condition is true;
else
    code to be executed if condition is false;
```

For-statement

- Sintassi praticamente uguale

```
for ($i=1; $i<=5; $i++) {
    echo "The number is " . $i . "<br>";
}
```

- Il for può essere usato per stampare contenuto derivante da un array. Contrariamente al foreach possiamo muoverci con maggiore libertà (possiamo stabilire da dove partire e dove arrivare, se muoverci in un senso o in un altro…)

```
$people = array(
```

```
    array('name' => 'Kalle', 'salt' => 856412),
    array('name' => 'Pierre', 'salt' => 215863)
);
for($i = 0, $size = sizeof($people); $i < $size; ++$i) {
    $people[$i]['salt'] = rand(000000, 999999);
}
```

Foreach-statement

- Il foreach consiste in una forma semplificata del for utilizzabile esclusivamente per scorrere gli array.
- Sono possibili due sintassi:
  - Una dove si pone in una variabile il contenuto dell'elemento dell'array
  - Una dove si pone la chiave dell'elemento dell'array, in aggiunta, in una seconda variabile.

```
foreach (array_expression as [&]$value)
    Statement

foreach (array_expression as $key => [&]$value)
    statement
```

- Attenzione: se non includiamo la & $value è solo una copia dell'array ed eventuali modifiche non saranno mantenute. Con la &, invece, $value si comporta come se fosse un alias dell'elemento dell'array.

Break-statement

- Stessa funzione del *break* in C++ (ossia uscire da un ciclo).
- La cosa interessante è che il break in PHP accetta un argomento (opzionale): possiamo indicare, attraverso un valore numerico da quanti cicli dobbiamo uscire.
- Supponiamo di essere all'interno di uno switch. Questo, a sua volta, si trova all'interno di un while.
  - Se poniamo `break 1;` usciremo dallo switch
  - Se poniamo `break 2;` , sempre all'interno dello switch, usciremo sia dallo switch che dal while.

- **Esempio**:

```
$i = 0;
while (++$i) {
    switch ($i) {
        case 5:
        echo "At 5<br>";
        break 1; /* Exit only the switch. */

        case 10:
        echo "At 10; quitting<br>";
        break 2; /* Exit the switch and the while. */

        default:
        break;
    }
}
```

Continue-statement

- Stessa funzione del continue in C++ (passare all'iterazione successiva di un ciclo).
- Le stesse cose dette per il break-statement (l'argomento numerico opzionale) valgono anche per il continue.
- **Esempio**:

```
$i = 0;
while ($i++ < 5) {
    echo "Outer<br>";
    while (1) {
        echo "  Middle<br>";
        while (1) {
            echo "  Inner<br>";
            continue 3;
        }
        echo "This never gets output.<br>";
    }
```

```
        echo "Neither does this.<br>";
    }
```

Sintassi alternativa per gli statement appena introdotti

- Sono presenti sintassi alternative per `if`, `while`, `for`, `foreach` e `switch`.
- Le parentesi graffe non sono presenti: si pone il nome del costrutto e i due punti, si chiude con lo statement `endNOMECOSTRUTTO;`

```
$a=6;
if ($a == 5):
    echo "a equals 5";
    echo "...";
elseif ($a == 6):
    echo "a equals 6";
    echo "!!!";
else:
    echo "a is neither 5 nor 6";
endif;
```

Include e require

- La `include` e la `require` permettono di includere file all'interno di altri file.
- Il file, oltre ad essere incluso, viene anche valutato.
- I file sono inclusi attraverso il percorso indicato come parametro delle istruzioni. Se non si pone un percorso in particolare sarà utilizzato quello presente nella *include_path*.
  Attenzione: il parametro non può essere un URL ASSOLUTO, non ha assolutamente senso. Se ci troviamo in un file e dobbiamo includere un qualcosa raggiungibile solo tornando indietro nelle cartelle (a partire dalla cartella del file) dobbiamo utilizzare "../"
  Esempio: ../../cartella/robadaincludere.php (vado indietro di due cartelle rispetto a quella del file dove sto lavorando)

- Se il file non viene trovato nella *include_path* si andrà a verificare nella directory dello script chiamante e in quella del lavoro corrente prima di produrre un fallimento.
  - La *include*, in caso di fallimento, emette una *warning*;
  - la *require*, invece, genera un errore *fatale*.

- **Esempio**:

vars.php
```
<?php
$color = 'green';
$fruit = 'apple';
?>
```

test.php
```
<?php
echo "A $color $fruit"; // A
include 'vars.php';
echo "A $color $fruit"; // A green apple
?>
```

- È possibile eseguire un `return-statement` all'interno di un file chiuso per terminare la processazione in quel file. È possibile restituire anche valori! Se non si restituiscono valori la funzione restituisce un booleano che indica il successo o meno dell'inclusione.

return.php
```
<?php
$var = 'PHP';
return $var;
?>
```

noreturn.php

```php
<?php
$var = 'PHP';
?>
```

testreturns.php

```php
<?php
$foo = include 'return.php';
echo "$foo<br>"; // prints 'PHP'
$bar = include 'noreturn.php';
echo $bar; // prints 1 (the include was successful)
?>
```

Include_once e require_once

- Le istruzioni include_once() e require_once() svolgono lo stesso lavoro delle istruzioni viste poco fa.
- L'unica differenza è che in caso di inclusione già avvenuta di un file non si procederà a nuova inclusione.

- La cosa è utile per evitare inclusioni annidate, fonti di loop e quindi di errori

Funzioni PHP

- Le funzioni in PHP sono definite in modo molto simile a quanto visto in Javascript: gli argomenti e l'eventuale valore da restituire non devono essere associati ad un tipo.

```php
<?php
function foo($arg_1, $arg_2, /* ..., */ $arg_n) {
    //….
    echo "Example function.\n";
    return $retval;
}
?>
```

- Le funzioni in PHP non necessitano di essere definite prima della chiamata.

- L'unico caso in cui l'ordine conta si ha con le cosiddette **funzioni definite condizionalmente**. Precisamente intendiamo funzioni dichiarate solo se certe condizioni sono soddisfatte. È ovvio che finchè queste condizioni non saranno soddisfatte la funzione non potrà essere chiamata. Vediamo il seguente esempio:

```php
<?php
$makefoo = true;

/* We can't call foo() from here since it doesn't exist yet, but we can
call bar() */

bar();
if ($makefoo) {
    function foo() {
        echo "I don't exist until program execution reaches me.\n";
    }
}

/* Now we can safely call foo() since $makefoo evaluated to true */
if ($makefoo)
    foo();

function bar() {
    echo "I exist immediately upon program start.\n";
}
?>
```

- Possiamo avere **<u>funzioni annidate</u>**, cioè funzioni all'interno di altre funzioni. Le funzioni incluse all'interno di altre funzioni non possono essere eseguite finché non saranno chiamate le funzioni più esterne.

```
<?php
function foo(){
    function bar() {
        echo "I don't exist until foo() is called.\n";
    }
}

/* We can't call bar() yet since it doesn't exist. */

foo();

/* Now we can call bar(), foo()'s processesing has made it accessible. */
bar();
?>
```

<u>Global scope, overloading e nomi di funzioni</u>

- Le funzioni e le classi in PHP hanno un **global scope**: possono essere chiamate al di fuori di una funzione nonostante siano state definite all'interno di una funzione. L o stesso ragionamento è valido a parti invertite.
- *L'overloading* di funzioni non è supportato, così come non è possibile rimuovere o ridefinire funzioni precedentemente dichiarate.
- I nomi delle funzioni sono *case-insensitive*.

<u>Passaggio degli argomenti</u>

- Argomenti passati per valore:
  - **Esempio 1**:

```
<?php
$arr = array(2,3);
takes_array($arr); \\ 2 + 3 = 5
takes_array($arr); \\ 2 + 3 = 5

function takes_array($input) {
     echo "$input[0] + $input[1] = ", $input[0]+$input[1], "<br>";
     $input[0]=10;
}
?>
```

  - **Esempio 2**:

```
<?php
function add_some_extra($string) {
     $string .= 'and something extra.';
}
$str = 'This is a string, ';
add_some_extra($str);

echo $str;  // outputs 'This is a string, '
?>
```

- Argomenti passati per riferimento
  - **Esempio 1**:

```
<?php
$arr=array(2,3);
takes_array($arr); \\ 2 + 3 = 5
takes_array($arr); \\ 10 + 3 = 13

function takes_array(&$input) {
     echo "$input[0] + $input[1] = ", $input[0]+$input[1], "<br>";
     $input[0]=10;
}
```

```
?>
```

- **Esempio 2**:

```
<?php
function add_some_extra(&$string) {
     $string .= 'and something extra.';
}
$str = 'This is a string, ';
add_some_extra($str);

echo $str;  // outputs 'This is a string, and something extra.'
?>
```

- Argomenti default
  - **Esempio**:

```
<?php
function makecoffee($type = "cappuccino") {
     return "Making a cup of $type. <br>";
}
echo makecoffee();  \\ Making a cup of cappuccino.
echo makecoffee(null);  \\ Making a cup of .
echo makecoffee("espresso");  \\ Making a cup of espresso.
?>
```

  - Il valore di default deve essere un'espressione costante: non una variabile, una funzione o un membro di una classe.

  - Ovviamente tutti gli argomenti default devono essere posti in fondo alla lista degli argomenti di una funzione: porre argomenti default all'inizio, prima di argomenti che devono essere per forza inizializzati, ci obbliga a indicare dei valori.

```
<?php
function makeyogurt($type = "acidophilus", $flavour) {
     return "Making a bowl of $type $flavour.\n";
}
echo makeyogurt("raspberry"); // won't work as expected
?>
```

Warning: *Missing argument 2 for makeyogurt(), called in c:\tia\www\php34.php on line 16 and defined in c:\tia\www\php34.php on line 11*

Funzioni con lista di argomenti a lunghezza variabile

- PHP4 permette di gestire funzioni con numero di argomenti variabile. La cosa può essere fatta attraverso le seguenti funzioni:
  - `func_get_arg(int)`, ottengo un argomento in particolare dato un certo indice.
  - `funct_get_args()`, ottengo un array con tutti gli argomenti.
  - `func_num_args()`, ottengo il numero degli argomenti della funzione.

- **Esempio**:

```
<?php
function foo() {
     $numargs = func_num_args();
     echo "Number of arguments: $numargs<br>";
     $arg_list = func_get_args();
     for ($i = 0; $i < $numargs; $i++) {
          echo "Argument $i is: " . $arg_list[$i] . "<br>";
     }
}
foo(1, 2, 3);
```

```
// Number of arguments: 3
// Argument 0 is: 1
// Argument 1 is: 2
// Argument 2 is: 3
?>
```

Valori restituiti

- Le funzioni possono restituire valori sfruttando il return-statement.
- Possiamo restituire qualunque cosa, incluse funzioni.

```
<?php
function small_numbers() {
     return array (0, 1, 2);
}
$arr = small_numbers();
print_r($arr); // Array ( [0] => 0 [1] => 1 [2] => 2 )
?>
```

Esecuzione di funzioni a partire da una variabile

- Se un nome di variabile è accompagnato da delle parentesi PHP cercherà di eseguire una funzione avente lo stesso nome.

```
<?php
function foo() {
     echo "In foo()<br>";
}
function bar($arg = '') {
     echo "In bar(); argument was '$arg'.<br>";
}
function echoit($string) {
     echo $string;
}

$func = 'foo';
$func(); // This calls foo()
$func = 'bar';
$func('test'); // This calls bar()
$func = 'echoit';
$func('test'); // This calls echoit()
?>
```

Built-in Functions

- In PHP esistono più di 700 funzioni built-in
- Le funzioni sono organizzate in categorie, precisamente:
  - Funzioni che influenzano il comportamento del PHP
  - Funzioni per la manipolazione di formati audio
  - Funzioni per servizi di autenticazione
  - Funzioni per gestire date e ore
  - Funzioni per la compressione e gli archivi

Funzioni per le date

- `date($format, $timestamp)`
  restituisce una stringa formattata secondo il formato indicato nel primo parametro. Se il secondo parametro non viene indicato si restituisce la data attuale. Il formato della data può essere stabilito attraverso dei caratteri. Dopo gli esempi è presente una tabella copiata da PHP.net contenente i caratteri utilizzabili nel format.

```
<?php
echo date("F j, Y, g:i a") . "<br>"; // January 7, 2011, 1:02 pm
echo date("m.d.y") . "<br>"; // 01.07.11
echo date("j, n, Y") . "<br>"; // 7, 1, 2011
echo date("Ymd") . "<br>"; // 20110107
echo date("F j, Y, g:i a", strtotime("10 September 2000")) . "<br>";
// September 10, 2000, 12:00 am
echo date("m.d.y") . "<br>"; // 01.07.11
echo date("F j, Y, g:i a", strtotime("+1 day")) . "<br>";
// January 8, 2011, 1:02 pm
?>
```

Character	Description	Example returned values
	Day	
d	Day of the month, 2 digits with leading zeros	01 to 31
D	A textual representation of a day, three letters	Mon through Sun
j	Day of the month without leading zeros	1 to 31
l	A full textual representation of the day of the week	Sunday through Saturday
N	ISO-8601 numeric representation of the day of the week (added in PHP 5.1.0)	1 (for Monday) through 7 (for Sunday)
S	English ordinal suffix for the day of the month, 2 characters	st, nd, rd or th. Works well with j
w	Numeric representation of the day of the week	0 (for Sunday) through 6 (for Saturday)
z	The day of the year (starting from 0)	0 through 365
	Week	
W	ISO-8601 week number of year, weeks starting on Monday	Example: 42 (the 42nd week in the year)
	Month	
F	A full textual representation of a month, such as January or March	January through December
m	Numeric representation of a month, with leading zeros	01 through 12
M	A short textual representation of a month, three letters	Jan through Dec
n	Numeric representation of a month, without leading zeros	1 through 12
t	Number of days in the given month	28 through 31
	Year	

L	Whether it's a leap year	1 if it is a leap year, 0 otherwise.
o	ISO-8601 week-numbering year. This has the same value as Y, except that if the ISO week number (W) belongs to the previous or next year, that year is used instead. (added in PHP 5.1.0)	Examples: 1999 or 2003
Y	A full numeric representation of a year, 4 digits	Examples: 1999 or 2003
y	A two digit representation of a year	Examples: 99 or 03
Time		
a	Lowercase Ante meridiem and Post meridiem	am or pm
A	Uppercase Ante meridiem and Post meridiem	AM or PM
B	Swatch Internet time	000 through 999
g	12-hour format of an hour without leading zeros	1 through 12
G	24-hour format of an hour without leading zeros	0 through 23
h	12-hour format of an hour with leading zeros	01 through 12
H	24-hour format of an hour with leading zeros	00 through 23
i	Minutes with leading zeros	00 to 59
s	Seconds with leading zeros	00 through 59
u	Microseconds (added in PHP 5.2.2). Note that date() will always generate 000000 since it takes an int parameter, whereas DateTime::format() does support microseconds if DateTime was created with microseconds.	Example: 654321
v	Milliseconds (added in PHP 7.0.0). Same note applies as for u.	Example: 654
Timezone		
e	Timezone identifier (added in PHP 5.1.0)	Examples: UTC, GMT, Atlantic/Azores
I	Whether or not the date is in daylight saving time	1 if Daylight Saving Time, 0 otherwise.
O	Difference to Greenwich time (GMT) without colon between hours and minutes	Example: +0200
P	Difference to Greenwich time (GMT) with colon between hours and minutes (added in PHP 5.1.3)	Example: +02:00
T	Timezone abbreviation	Examples: EST, MDT ...
Z	Timezone offset in seconds. The offset for timezones west of UTC is always negative, and for those east of UTC is always positive.	-43200 through 50400

- `time()`
  restituisce l'orale attuale in formato UNIX (numero di secondi passati dalla Epoch)
  Esempio di output: 1605112269

- `strtotime($time)`
  permette di convertire una data di qualunque formato in un formato UNIX (cioè restituisco un intero che consiste nel numero di secondi passati dalla Epoch). Restituisce false in caso di fallimento nella conversione.

- `getdate($timestamp = time())`
  funzione che restituisce un array associativo contenente le informazioni della timestamp, o dell'ora attuale se non viene dato in ingresso nessun timestamp. Vediamo un esempio dove sono chiare le chiavi utilizzabili:

```
<?php
$array = getdate();
var_dump($array);
?>
```

Risultato:

```
array(11) {
  ["seconds"]=>
  int(17)
  ["minutes"]=>
  int(28)
  ["hours"]=>
  int(17)
  ["mday"]=>
  int(11)
  ["wday"]=>
  int(3)
  ["mon"]=>
  int(11)
  ["year"]=>
  int(2020)
  ["yday"]=>
  int(315)
  ["weekday"]=>
  string(9) "Wednesday"
  ["month"]=>
  string(8) "November"
  [0]=>
  int(1605112097)
}
```

PHP Classes

- La sintassi è la stessa di Java
- La visibilità degli elementi può essere definita attraverso le keyword `public`, `protected` o `private`. Il loro significato è lo stesso visto in C++.
- Le dichiarazioni possono includere inizializzazioni, ma solo basandosi su valori costanti.
- Metodi dichiarati senza una esplicita visibilità sono definiti come public.
- PHP5 permette la creazione di costruttori e distruttori.

```
<?php
class MyDestructableClass {
     private $name="Default: ";

     function __construct() {
          print "In constructor\n";
          $this->name .= "MyDestructableClass";
     }

     function __destruct() {
          print "Destroying " . $this->name . "\n";
     }
}
$obj = new MyDestructableClass;
?>
```

Output:

```
In constructor
Destroying Default: MyDestructableClass
```

- Per ragioni di compatibilità se non si individua un costruttore `__construct()` si andrà ad eseguire una funzione che ha per nome quello della classe (quest ultimo metodo arcaico).
- `$this` non è un puntatore ma un riferimento.

- Se non si vuole ricevere errori quando utilizziamo l'operatore new basterà porre la chiocciola prima della keyword.

- **Esempio classico**: la pila (mettetevi nel capo il concetto di pila, serve a Reti logiche e servirà a Calcolatori)

```
<?php
class Stack {
     private $top, $vett;
     function Stack() {
          print "In Stack\n";
          $this->top =-1;
     }

     function push($elem) {
          $this -> vett[++$this->top]=$elem;
     }

     function pop() {
          if ($this->top>-1)
               return $this -> vett[$this->top--];
          return null;
     }
}
$mystack = new Stack;
$mystack->push(15);
echo $mystack->pop()."<br>";
echo $mystack->pop()."<br>";
?>
```

Output: In Stack 15

  - Abbiamo due variabili private: $top e $vett. Queste non possono essere utilizzate al di fuori della classe
  - Le funzioni non presentano etichette di visibilità particolari, quindi sono pubbliche e chiamabili dall'esterno della funzione.

- **Ereditarietà**:
  - I meccanismi dell'ereditarietà visti in C++ sono validi anche in PHP.
  - Possiamo definire una classe figlia di un'altra utilizzando la keyword extends

```
<?php
class Foo {
     public function printItem($string) {
          echo 'Foo: ' . $string . "<br>";
     }

     public function printPHP() {
          echo 'PHP is great.' . "<br>"; // Marco Lampis è in arresto cardiaco
     }
}

class Bar extends Foo {
     public function printItem($string) {
          echo 'Bar: ' . $string . "<br>";
     }
}

$foo = new Foo();
$bar = new Bar();
$foo->printItem('baz'); // Output: 'Foo: baz'
$foo->printPHP(); // Output: 'PHP is great'
$bar->printItem('baz'); // Output: 'Bar: baz'
$bar->printPHP(); // Output: 'PHP is great'
?>
```

- La classe `Bar` è figlia della classe `Foo`.
- Posso chiamare le funzioni definite nella Foo tenendo conto di eventuali funzioni ridefinite, come in questo caso con la `printItem`.

- **Valori costanti**:
  - Possono essere definite costanti relativamente a una classe (non a singole istanze).
  - La chiamata della costante avviene attraverso l'operatore di risoluzione di visibilità.

```
<?php
class MyClass {
     const CONST_VALUE = 'A constant value';
}
echo MyClass::CONST_VALUE;
?>
```

- **Ulteriori usi dell'operatore di risoluzione di visibilità**:
  - L'operatore di risoluzione di visibilità può essere utilizzato per richiamare funzioni istanza di una classe padre.

```
<?php
class MyClass {
     protected function myFunc() {
          echo "MyClass::myFunc()<br>";
     }
}
class OtherClass extends MyClass { // Override parent's definition
     public function myFunc() { // But still call the parent fnct
          parent::myFunc();
          echo "OtherClass::myFunc()<br>";
     }
}
$class = new OtherClass();
$class->myFunc();
//Chiamo MyClass::myFunc()
//Chiamo anche OtherClass::myFunc()
?>
```

- **Keyword *static***:
  - La keyword `static` permette di dichiarare variabili rendendole accessibili senza un istanziamento della classe (sono elementi non collegati a una singola istanza, ma alla classe)
  - Ovviamente un membro dichiarato come `static` non può essere raggiunto a partire da un oggetto istanza della classe.

```
<?php
class Foo {
     public static $my_static = 0;
     public function staticValue() {
          return ++self::$my_static;
     }
}
class Bar extends Foo {
     public function fooStatic() {
          return ++parent::$my_static;
     }
}
print Foo::$my_static . "<br>"; //0

$foo = new Foo();
print $foo->staticValue() . "<br>"; //1
```

```
print $foo->my_static . "<br>"; // Undefined "Property" my_static

// $foo::my_static is not possible

print Bar::$my_static . "<br>"; //1
$bar = new Bar();
print $bar->fooStatic() . "<br>"; //2
?>
```

Object serialization

- La serializzazione permette di esportare le informazioni relative all'istanza di una classe.
- Possiamo gestirla attraverso due funzioni:
  - `serialize()`
    Restituisce una stringa contenente uno stream di byte rappresentazione di un qualunque valore ospitabile in PHP. Se passiamo un oggetto la serialize terrà conto di tutte le variabili presenti all'interno dell'oggetto.

  - `unserialize()`
    utilizza una stringa generata con serialize per rigenerare l'oggetto originale. La classe di cui l'oggetto è istanza deve essere nota all'interno del codice.

  - **Osservazioni**:
    - Il PHP, dopo aver chiamato *serialize*, verifica se la classe coinvolta ha una funzione membro denominata `__sleep`. Se presente la esegue permettendo eventuali azioni prima della serializzazione.
    - Il PHP, dopo aver chiamato *unserialize*, verifica se la classe coinvolta ha una funzione membro denominata `__wakeup`. Se presente la esegue.

  - **Esempio**:

```
<?php
// classa.inc: definizione della classe (obbligatoria, la struttura deve essere nota)
class A {
     public $one = 1;

     public function show_one() {
         echo $this->one;
     }
}

// page1.php:
include("classa.inc");

$a = new A;
$s = serialize($a);
// store $s somewhere where page2.php can find it.
file_put_contents('store', $s);

// page2.php:
// this is needed for the unserialize to work properly.
include("classa.inc");

$s = file_get_contents('store');
$a = unserialize($s);

// now use the function show_one() of the $a object.
$a->show_one();
?>
```

Gestione dei forms lato server

- Ogni valore impostato attraverso i controlli del form è accessibile dallo script PHP.

- Per recuperare le informazioni dei form si utilizzano i seguenti array globali associativi:
  - `$_GET`, array associativo di variabili passate allo script attraverso i parametri dell'URL
  - `$_POST`, array associativo di variabili passate allo script attraverso il metodo HTTP POST
  - `$_REQUEST`, array associativo che contiene quanto presente in $_GET, $_POST e $_COOKIE

Esempio di invio con metodo POST

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Form Example</title>
  </head>

  <body>
    <form action="elab.php" method="post">
      <p>
      Name:<br>
      <input type="text" name="username"><br>
      E-mail:<br>
      <input type="text" name="email"><br>
      <input type="submit" name="submit" value="SUBMIT">
      </p>
    </form>
  </body>
</html>
```

In `elab.php` possiamo richiamare i valori dei controlli utilizzando il seguente codice

```
<?php
echo $_POST['username']; // value1
echo $_POST['email']; // value2
echo $_POST['submit']; // value3
// oppure
echo $_REQUEST['username']; // value1
?>
```

Esempio di invio con metodo GET

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Form Example</title>
  </head>

  <body>
    <form action="elab.php" method="get">
      <p>
      Name:<br>
      <input type="text" name="username"><br>
      E-mail:<br>
      <input type="text" name="email"><br>
      <input type="submit" name="submit" value
      </p>
    </form>
  </body>
</html>
```

Sarà aperta la seguente pagina:

`elab.php?username=value1&email=value2&submit=value3`

In `elab.php` possiamo richiamare i valori dei controlli utilizzando il seguente codice

```
<?php
echo $_GET['username']; // value1
echo $_GET['email']; // value2
echo $_GET['submit']; // value3
// oppure
echo $_REQUEST['username']; // value1
?>
```

Validazione form

- Gli input degli utenti devono essere sempre validati lato client quando possibile: la validazione lato client è più veloce e riduce le richieste al server.
- La validazione lato server va considerata se l'input dell'utente deve essere inserito all'interno di un database.
- Una buona regola per validare una form sul server è impostare come action la pagina stessa in cui è posto il form. Non serve indicare il valore dell'attributo in questo caso.

- Le informazioni inviate da un form col metodo GET sono visibili a tutte ma il limite relativo alla quantità di informazioni inviabili è di 100 caratteri.
- Le informazioni inviate da un form col metodo POST sono invisibili agli altri e non presentano un limite di dimensione. Per gli amanti dell'ovvio: con questo metodo non è possibile salvare la pagina visitata nei segnalibri.

Metodo alternativo per accedere alle variabili GET/POST/Cookie

Un metodo alternativo consiste nell'utilizzo della seguente funzione

- `import_request_variables($types, $prefix);`
  Presenta due parametri
  - `types`: specifichiamo quali variabili vogliamo importare. I valori possibili sono i seguenti
    - G per GET
    - P per POST
    - C per Cookie

    l'argomento non è case-sensitive.
  - `prefix`: facoltativo, permette di stabilire un prefisso per i nomi delle variabili che andremo a creare.

  Come al solito le cose si capiscono meglio facendo degli esempi: supponiamo di aprire la pagina `elab.php?username=Frax&email=tipiacerebbe`. Usiamo la funzione nel codice:

```
<?php
import_request_variables('g','datiutente_');
echo $datiutente_username; //"Frax"
echo $datiutente_email; //"tipiacerebbe"
?>
```

Cookie

- I cookie consistono in piccoli pezzi di dati inviati dal server di un sito e ospitati all'interno del browser utilizzato dall'utente. Tutto questo avviene mentre l'utente sta visitando il sito.

- Il server che ospita il sito specifica quali cookie devono essere ospitati dal browser inviando un intestazione http chiamata Set-Cookie, avente la seguente forma
  `Set-Cookie: value[; expires=date][; domain=domain][;path=path][; secure]`

- Quando un cookie è presente, e le impostazioni del browser lo permettono, il suo valore può essere inviato al server ad ogni successiva richiesta. Il valore del cookie è ospitato in un'intestazione HTTP detta Cookie che contiene solo il valore del cookie senza ulteriori elementi.
  `Cookie: value`

- **Osservazione**: i cookie devono essere inviati prima che venga espresso un qualunque output dal nostro codice (cosa valida per qualunque intestazione http).

- Per gestire i cookie utilizziamo la seguente funzione
  `setCookie($name, $value, $expire, $path, $domain, $secure, $httponly)`

  - `name`: richiesto, nome del cookie
  - `value`: facoltativo, valore del cookie

  - `expire`: facoltativo, data di scadenza del cookie. A partire da quella data il cookie non sarà più spedito al server e il browser potrà eliminarlo. Se non si specifica il valore il cookie sarà terminato dopo la chiusura del browser.

  - `path`: facoltativo, indica il percorso del cookie sul server. Indicare un path significa limitare l'accesso del cookie a specifiche directories e subdirectories del sito. Il valore di default è la directory dove ci troviamo, quella dove viene settato il cookie.

  - `domain`: facoltativo, indica i domini a cui il cookie deve essere inviato. Di default si indica l'hostname della pagina che sta inizializzando il cookie. Indicare il domain permette di ampliare il numero di domini a cui il valore del cookie sarà spedito.

        - Grandi network come Yahoo presenta un certo numero di siti aventi come dominio `name.yahoo.com`
        - Impostando il dominio indico il valore di un cookie valido per tutti questi siti appartenenti alla galassia di Yahoo (indico solo `.yahoo.com`)

    - `secure`: facoltativo, booleano che specifica se il cookie debba essere accessibile o meno solo mediante il protocollo HTTPS (cioè solo con una connessione sicura). Il valore di default è false.

    - `httponly`: facoltativo, booleano con cui stabilisco se il cookie debba essere accessibile o meno solo mediante il protocollo http (il cookie non sarà accessibile a linguaggi di scripting, cosa che permette di ridurre furti di identità attraverso attacchi XSS). Il valore di default è false.

- **Esempio di intestazione restituita da un programma in PHP:**

```
HTTP/1.1 200 OK
Date: Fri, 04 Feb 2000 21:03:38 GMT
Server: Apache/1.3.9 (UNIX) PHP/4.0b3
Set-Cookie: name=xyz; expires=Friday, 04-Feb-07 22:03:38 GMT;
path=/; domain=tutorialspoint.com
Connection: close
Content-Type: text/html
```

- **Esempio di intestazione inviata da un browser al server**:

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)
Host: zink.demon.co.uk:1126
Accept: image/gif, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: name=xyz
```

- **Esempio di codice**:

```
<?php
$value = 'something from somewhere';
setcookie("TestCookie", $value, time()+(60*60), "/~rasmus/",
".example.com", 1);
echo $_COOKIE[“TestCookie”]; // Array associativo da cui posso recuperare un cookie dato il nome
?>
```

    - Abbiamo impostato un cookie chiamato TestCookie e avente come valore il contenuto della variabile $value.
    - Il cookie scadrà tra un’ora.
    - Il cookie è disponibile sul dominio example.com e su tutti i sottodomini (attenzione al punto prima del dominio).
    - Il cookie sarà disponibile solo in presenza di una connessione sicura (protocollo HTTPS)

- **Come si cancellano i cookie?** Stessa strategia vista coi Cookie in Javascript: attraverso la `setCookie` imposto una `$expire` che indica una data antecedente a quella attuale.

```
<?php
setcookie("TestCookie", "", time()- 60, "/","", 0);
?>
```

Sessions

- Una sessione ci permette di preservare certi dati in accessi successivi. Questi dati, contrariamente ai cookie, non sono conservati nel computer dell'utente.
- Al visitatore del nostro sito web viene assegnato un id univoco, detto session id. Questo può essere memorizzato in un cookie o propagato all'interno di un URL.
- Le sessioni permettono di creare un numero arbitrario di variabili da preservare nel corso di più richieste.

- **Cosa fa il PHP di preciso?**
  - Quando l'utente visita il sito PHP verifica automaticamente (se `session.auto_start` è impostato ad 1) o manualmente (se richiesto esplicitamente con la `session_start()`) se è stato spedito un id di sessione con la richiesta.
  - Se ciò avviene quanto salvato precedentemente viene recuperato.

- **Scopo principale**: per quanto riguarda questo corso lo scopo principale delle variabili di sessione consiste nella creazione di un sistema di login, cioè richiedere l'inserimento di dati (solitamente username e password) per accedere ad aree riservate del nostro sito. I dati da inserire possono essere salvati in due modi.
  - **Salvati in variabili PHP**. I dati possono essere modificati solo dal programmatore: segue che la strategia è conveniente solo quando pochi utenti devono accedere all'area riservata e questi non hanno bisogno di modificare le proprie credenziali.
  - **Salvati in una tabella di un database**. La scelta è conveniente quando più utenti devono accedere all'area riservata. Porre i dati in un database permette di creare una pagina di registrazione per i nuovi utenti e una di modifica password per chi vuole modificare le proprie credenziali.

- **Cosa determina se abbiamo fatto login?** Una variabile di sessione.

- **Quali pagine dobbiamo creare per gestire un sistema di login?**
  - Una pagina per fare login. Si modifica la variabile di sessione per indicare il login effettuato.
  - Una pagina col contenuto riservato. Si verifica la variabile sessione per capire se l'utente ha fatto login o no. Se ha fatto login si mostra il contenuto, altrimenti si stampa un errore o si reindirizza al login.
  - Una pagina per fare logout. Si modifica la variabile di sessione per indicare che l'utente non ha fatto login.

- **Principio da tenere nel campo**: una pagina non è nascosta se ci limitiamo a bloccare l'accesso solo alle pagine da percorrere per raggiungerla. Un utente può raggiungere la pagina bypassando l'area "bloccata" se conosce l'URL. **OGNI SINGOLA PAGINA DEVE ESSERE PROTETTA.**

  - **Codici base da cui partire:**
    - login.php

```
<?php
session_start();

if($_POST['username'] == 'Frax' && $_POST['password'] == 'ciaomarco') {
    $_SESSION['logged'] = 1;

    // Login effettuato, decidiamo noi cosa fare
    // Solitamente si reindirizza all'area riservata
    // adesso consultabile
}
?>
```

    - page.php

```
<?php
session_start();

if($_SESSION['logged'] == NULL) {
```

```
    // Non abbiamo fatto login
    // Possiamo stampare un messaggio di errore
    // oppure reindirizzare l'utente sulla pagina di login
    // abbiamo già visto come si fa (o con Javascript o con PHP)
}
else {
    // Il contenuto di questa pagina sarà mostrato solo ad utenti che hanno fatto login
    echo 'Ciao, bentornato!';
}
?>
```

- logout.php

```
<?php
session_start();

unset($_SESSION['logged']);

// Applicare l'unset a questo valore comporta il logout

// Adesso decidiamo noi cosa fare....
// Solitamente si reindirizza alla pagina iniziale del sito
// o alla pagina di Login.
?>
```

PHP e MySQL

- PHP permette di interfacciarsi con un database e svolgere interrogazioni.
- I metodi per connettersi a un database sono diversi: uno dei principali è la classe `PDO` (acronimo di PHP Data Objects), le cui istanze rappresentano connessioni tra il PHP e il server di un database. PDO supporta dodici databases differenti.

- L'unica cosa che guarderemo in questo corso è la classe `mysqli`[1], le cui istanze rappresentano connessioni tra il PHP e un database MySQL.

Cose interessanti dalla classe `mysqli`

- `mysqli::__construct(`**`$host, $username, $passwd, $dbname`**`, $port, $socket)`
  Il costruttore permette di aprire una connessione col database. Ci interessano i primi quattro parametri:
  - $`host`, si indica l'hostname o un indirizzo IP (l'identificativo dell'`host` che ospita il database, se il valore è nullo o uguale a 'localhost' si intende come host quello locale);
  - $`username`, nome utente;
  - $`passwd`, password;
  - $`dbname`, nome del database su cui vogliamo lavorare.

- `mysqli::$connect_error`
  stringa con la descrizione dell'ultimo errore di connessione

- `mysqli::$connect_errno`
  codice numerico dell'errore dall'ultima chiamata di connessione

- `mysqli::$host_info`
  stringa contenente il tipo di connessione usata. La stringa presenta la seguente forma
  `HOST via TIPO_CONNESSIONE`

Esempio:
```
<?php
$mysqli= new mysqli($nomeHost, $nomeUtente, $password, $db);
if ($mysqli->connect_error) { // Se è presente un errore ($mysqli->connect_error != NULL)
    die('Connect Error (' . $mysqli->connect_errno . ') '. $mysqli->connect_error);
}
echo 'Success... ' . $mysqli->host_info . "\n";
$mysqli->close();
?>
```

- `mysqli::select_db($database_name)`
  selezione del database su cui eseguire query (solitamente lo si indica con gli argomenti del costruttore)

Esempio:
```
$mysqli->select_db($nomeDb) or die ('Can\'t use pweb: ' . mysql_error());
```

- `mysqli::query($query)`
  funzione membro con cui si esegue l'interrogazione $query.
  - Per le interrogazioni `SELECT, SHOW, DESCRIBE, EXPLAIN` e altri SQL statements che restituiscono un result set la funzione restituisce un oggetto `mysqli_result` se ha successo, altrimenti *false*.
    - Necessario utilizzare delle funzioni per ottenere un array contenente i risultati.
    - Queste funzioni sono introdotte nella sezione successiva.
  - Per le query `INSERT, UPDATE, DELETE, DROP` e tutti gli altri SQL statements di cui ci interessa al più l'esito (quindi se l'operazione è andata a buon fine) si restituisce *true* o *false*.

[1] La *i* in fondo sta per *improved*.

- `mysqli::close()`
  chiusura della connessione al database precedentemente aperta.

- `mysqli::$affected_rows`
  numero di righe coinvolte nell'ultima operazione svolta (molto utile con UPDATE, DELETE, SELECT…)

- `mysqli::$errno`
  codice numerico dell'ultimo errore che si è manifestato con la chiamata di funzione più recente.

- `mysqli::$error`
  stringa con la descrizione dell'ultimo errore che si è manifestato con la chiamata di funzione più recente.

Cose interessanti dalla classe mysqli-result

- `mysqli_result::fetch_assoc()`
  Gestisce una riga dei risultati come un array associativo.

```
array(3) {
  ["id"]=>
  string(3) "118"
  ["f1"]=>
  string(5) "Marco"
  ["f2"]=>
  string(22) "Ciao Marco, come stai?"
}
```

- `mysqli_result::fetch_row()`
  Gestisce una riga dei risultati come un array numerico.

```
array(3) {
  [0]=>
  string(3) "118"
  [1]=>
  string(5) "Marco"
  [2]=>
  string(22) "Ciao Marco, come stai?"
}
```

- `mysqli_result::fetch_array($resulttype)`
  Gestisce una riga dei risultati come un array associativo, un array numerico o entrambi. Il tipo di gestione può essere indicato attraverso le seguenti costanti:
  - `MYSQLI_ASSOC`, array associativo (equivalente della `fetch_assoc()`)
  - `MYSQLI_NUM`, array numerico (equivalente della `fetch_row()`)
  - `MYSQLI_BOTH`, singolo array che consiste nell'unione dell'array associativo e di quello numerico.

  Il valore di default (`$resulttype` è opzionale) è `MYSQLI_BOTH`.

```
array(6) {
  [0]=>
  string(3) "118"
  ["id"]=>
  string(3) "118"
  [1]=>
  string(5) "Marco"
  ["f1"]=>
  string(5) "Marco"
  [2]=>
  string(22) "Ciao Marco, come stai?"
  ["f2"]=>
  string(22) "Ciao Marco, come stai?"
}
```

- `mysqli_result::fetch_all($resultype)`
  genera un array contenente le righe del result set. Relativamente al parametro valgono le stesse cose dette per `fetch_array`. L'unica differenza è che il valore di default è `MYSQL_NUM`.

```
array(115) {
  [0]=>
  array(3) {
    ["id"]=>
    string(3) "118"
    ["f1"]=>
    string(5) "Marco"
    ["f2"]=>
    string(22) "Ciao Marco, come stai?"
  }
  [1]=>
  array(3) {
    ["id"]=>
    string(3) "156"
    ["f1"]=>
    string(1) "F"
    ["f2"]=>
    string(4) "𝓕"
  }
  [2]=>
  array(3) {
    ["id"]=>
    string(3) "328"
    ["f1"]=>
    string(9) "pistolesi"
    ["f2"]=>
    string(35) "Lascia l'apparenza e cogli il senso"
```

Prima parte dell'array

- `mysqli_result::$lengths`
  array relativo alla lunghezza delle colonne della riga corrente del result set.

```
while($info = $query->fetch_array()) {
    echo '<pre>'; var_dump($query->lengths); echo'</pre>';
```

```
array(115) {
  [0]=>
  array(3) {
    ["id"]=>
    string(3) "118"
    ["f1"]=>
    string(5) "Marco"
    ["f2"]=>
    string(22) "Ciao Marco, come stai?"
  }
  [1]=>
  array(3) {
    ["id"]=>
    string(3) "156"
    ["f1"]=>
    string(1) "F"
    ["f2"]=>
    string(4) "𝓕"
  }
  [2]=>
  array(3) {
    ["id"]=>
    string(3) "328"
    ["f1"]=>
    string(9) "pistolesi"
    ["f2"]=>
    string(35) "Lascia l'apparenza e cogli il senso"
```

```
array(3) {
  [0]=>
  int(3)
  [1]=>
  int(5)
  [2]=>
  int(22)
}

array(3) {
  [0]=>
  int(3)
  [1]=>
  int(1)
  [2]=>
  int(4)
}

array(3) {
  [0]=>
  int(3)
  [1]=>
  int(9)
  [2]=>
  int(35)
}
```

Tre esempi di array $lengths se consideriamo i primi elementi dell'array ottenuto con fetch_all

- `mysqli_result::$num_rows`
  numero delle righe ottenute eseguendo un'interrogazione con `query()`

Esempio di codice per eseguire interrogazioni

- Come dice Pistolesi le vie del signore sono infinite. Vediamo come possiamo utilizzare queste funzioni.

- **Solitamente la via privilegiata è questa**:

```
$con = mysqli_connect(. . . . .);
if ($con->connect_error) {
  die('Connect Error (' . $con->connect_errno . ') '. $con->connect_error);
}

$query = $con->query("SELECT id,f1,f2 FROM frasi ORDER BY
num_triggeraggi DESC, id DESC") or die("<b>Errore</b>: ".$con->error);

while($info = $query->fetch_assoc()) {
    // contenuto da stampare
    // Ogni volta che chiamo la funzione passo all'elemento successivo
    // continuo finchè non avrò passato tutte le righe del result set
    echo $info["id"];
}
$con->close();
```

- **Si può lavorare anche così ma non ve lo consiglio:**

```
$con = mysqli_connect(. . . . .);
if ($con->connect_error) {
  die('Connect Error (' . $con->connect_errno . ') '. $con->connect_error);
}

$query = $con->query("SELECT id,f1,f2 FROM frasi ORDER BY
num_triggeraggi DESC, id DESC") or die("<b>Errore</b>: ".$$con->error);

$array = $query->fetch_all(MYSQLI_ASSOC);
foreach($array as $elemento_array) {
    // contenuto da stampare
    echo $elemento_array["id"];
}
$con->close();
```

Controllo dei valori inseriti

- Gli esempi posti precedentemente sono ottimi finché non iniziamo a eseguire interrogazioni dipendenti da variabili (soprattutto se il valore di queste variabili è indicato dall'utente che visita il sito).
- Filtrare queste variabili è vitale per due ragioni:
  - Evitare che caratteri speciali intacchino il corretto funzionamento della query
  - Evitare le cosiddette SQL injections

Caratteri fastidiosi: supponiamo di eseguire la seguente query

```
$lastname = "D'Annunzio";
$query="INSERT INTO Persons (LastName) VALUES ('$lastname')";
$result = $mysqli->query($query);
```

la variabile `$result` restituirà *false*: la query non può essere eseguita poiché sintatticamente errata. Quale testo abbiamo ottenuto ponendo all'interno della query la variabile `$lastname`?

```
echo $query; // "INSERT INTO Persons (LastName) VALUES ('D'Annunzio')"
```

Attenzione all'apice posto come apostrofo! L'apice dell'apostrofo viene considerato come la chiusura di un'espressione letterale (cioè il valore di `LastName`). La soluzione consiste nell'inserire un *backslash* prima dell'apostrofo.

```
$lastname = "D\'Annunzio";
```

in PHP esiste la funzione `addslashes($string)` che permette l'aggiunta di backslash prima dei seguenti caratteri:

- apici
- doppi apici
- backslash

Quanto proposto è un inizio ma non è sufficiente!

SQL Injections: le SQL injections consistono in iniezioni di codice col chiaro obbiettivo di ottenere informazioni private (normalmente non accessibili) o modificare il database. Nella scrittura del codice PHP dobbiamo prevenire questi attacchi. Prendiamo gli esempi delle diapositive di Marcelloni: la query che vogliamo eseguire è la seguente

```
$miaQuery = "SELECT COUNT(*) FROM utenti WHERE username=' "
.$_POST['username']. " ' AND password=' " .$_POST['password']. " ' ";
```

il contenuto della query dipende dalle variabili POST, indicate dall'utente. Una query del genere può essere utilizzata in un login: l'utente vuole accedere a un'area riservata inserendo le sue credenziali. Non filtrare queste variabili significa autorizzare i visitatori a modificare la query: un malintenzionato potrebbe manipolarla in modo tale da ottenere l'accesso all'account dell'amministratore.

Vediamo due esempi di manipolazione: in entrambi teniamo conto della precedenza dell'operatore AND rispetto all'operatore OR.

Input the following data:

username GIANNI password ' OR ''='

```
SELECT COUNT(*)
WHERE username='GIANNI'
AND password= '' OR '' = '';

The query results to be:
SELECT COUNT(*) FROM utenti
WHERE username='GIANNI'
AND password= '' OR '' = '';
```

Noi non conosciamo la password, ma impostare la variabile $password così rende superfluo conoscerla.

L'espressione '' = '' è sempre vera.

```
SELECT COUNT(*)
WHERE username= '' OR 1=1 --'
AND password= '';
```

Anche in questo caso diventa superfluo conoscere la password. Attenzione ai trattini "--": in MySQL rappresentano l'inizio di un commento.
Segue che la parte della query relativa alla password diventerà un commento e non sarà considerata.
Inoltre l'espressione 1 = 1 è sempre vera!

Come risolviamo questi problemi? Filtriamo le variabili attraverso la funzione `mysqli_real_escape_string($string)`:

```
$username = mysqli_real_escape_string($_POST['username']);
$password = mysqli_real_escape_string($_POST['password']);
$miaQuery = "SELECT COUNT(*) FROM utenti WHERE username=' ".$username. "
' AND password=' ".$password. " ' ";
```

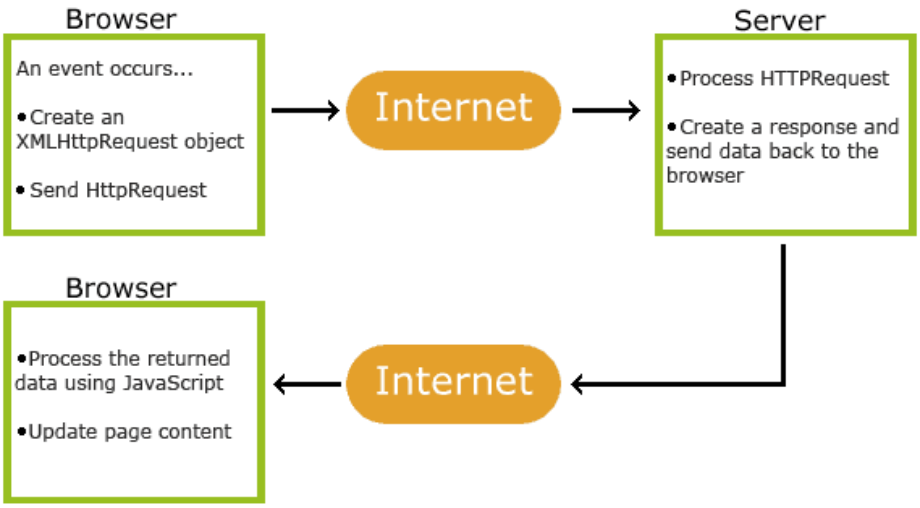
Perché non basta la `addslashes`? Dobbiamo tenere conto di tutte le *keywords* utilizzabili all'interno di una query. La funzione `mysqli_real_escape_string` fa escape su tutti i caratteri speciali da considerare.

AJAX[2]

- AJAX è acronimo di *Asynchronous JavaScript And XML*[3]. Non consiste in un linguaggio di programmazione autono, ma in un gruppo di tecniche utilizzate sul lato client per creare applicazioni web interattive.
- Attraverso AJAX le applicazioni web possono recuperare dati da un server in modo asincrono senza interferire con la visualizzazione e il comportamento della pagina esistente.

- **Su cosa si basa AJAX?**
  - Un oggetto built-in `XMLHttpRequest` (con cui richiediamo dati da un web server)
  - Codice Javascript e HTML DOM (per utilizzare i dati ottenuti dal web server)

- **Come funziona AJAX?**



  - Un evento occorre in una pagina web (per esempio il caricamento completato della pagina o il click di un bottone).
  - Si crea con Javascript un oggetto `XMLHttpRequest`
  - L'oggetto `XMLHttpRequest` creato invia una richiesta a un server.
  - Il server processa la richiesta e invia una risposta alla pagina web.
  - La risposta viene letta da Javascript.
  - Javascript esegue le istruzioni necessarie (per esempio l'aggiornamento del contenuto della pagina)

Oggetto XMLHttpRequest

- L'oggetto è utilizzato per inviare richieste HTTP o HTTPS direttamente a un web server e ottenere indietro la risposta direttamente nello script.
- Per ragioni di sicurezza i browser moderni consentono lo scambio di dati solo tra pagine su uno stesso server. Questo significa che la pagina dove è presente il codice Javascript e quella a cui facciamo la richiesta dovranno trovarsi nello stesso server.
- I dati ottenuti possono essere utilizzati per alterare il DOM del documento.

- **Funzioni offerte dall'oggetto XMLHttpRequest** (tolte cose non necessarie):

Funzione	Descrizione
new XMLHttpRequest()	Crea un nuovo oggetto XMLHttpRequest
abort()	Cancella l'attuale richiesta.
open(*method,url,async,user,psw*)	Specifica la richiesta. Presenta i seguenti argomenti: - *method*: il tipo di richiesta (GET o POST) - *url*: l'indirizzo del file da visitare (ricordarsi che deve essere sullo stesso server) - *async*: booleano, si indica se la richiesta deve essere gestita in modo asincrono o no - *user* e *psw:* opzionali, nome utente e password per aree riservate (cioè aree che richiedono nel corso di una transazione HTTP delle credenziali)
send()	Invio la richiesta al server (richieste di tipo GET)

[2] Il docente ha messo questo argomento nelle diapositive sul PHP.

[3] AJAX non è un nome proprio preciso. Per lo scambio di dati possiamo usare anche *plain text* e *JSON text*.
Il JSON sarà affrontato nei prossimi laboratori del corso.

send(*string*)	Invio la richiesta al server (richieste di tipo POST). L'argomento string contiene i valori inviati (nella forma `name1=value1&name2=value2 …`)
setRequestHeader(*header, value*)	Aggiunge intestazioni HTTP alla richiesta. Richiesti i seguenti parametri: - *header*: nome dell'header - *value*: valore dell'header

- **Proprietà dell'oggetto XMLHttpRequest**:

Proprietà	Descrizione
readyState	Restituisce lo stato della richiesta al server. I valori possibili sono i seguenti: 0. Richiesta non inizializzata 1. Connessione col server stabilita 2. Richiesta ricevuta 3. Richiesta in processazione **4. Richiesta conclusa e risposta disponibile**
onreadystatechange	Definisce una funzione che sarà eseguita quando la proprietà `readyState` cambia di valore.
responseText	Restituisce la risposta come una stringa.
responseXML	Restituisce la risposta come XML.
status	Restituisce lo status di una richiesta. I valori restituiti più ricorrenti sono: - ***200***: "OK" - ***403***: "Forbidden" - ***404***: "Not Found"
statusText	Restituisce la stessa cosa di `status`, ma in formato testuale.

Step necessari per creare una richiesta

1. **Creare un oggetto `XMLHttpRequest`**: il codice seguente gestisce problemi di compatibilità (come al solito il problema è Internet Explorer).

```
try {
  xmlHttp=new XMLHttpRequest();
}
catch (e) {
  try {
    xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
  }
  //IE (recent versions)
  catch (e) {
    try {
      xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
    //IE (older versions)
    catch (e) {
      window.alert("Your browser does not support AJAX!");
      return false;
    }
  }
}
```

   La cosa non è strettamente necessaria per i nostri progetti: non dobbiamo farli funzionare su Explorer.

2. **Scrivere un handler dell'evento per inviare le richieste per dati al server**. Vediamo le due strade possibili:
   - **Metodo GET**:
     // Creazione dell'handler. La richiesta è di tipo GET ed è rivolta alla pagina *data.php* presente sullo stesso server del nostro documento. La richiesta DEVE essere asincrona: dire questo significa permettere a Javascript di svolgere altre azioni mentre attende una risposta da parte del server. Porre il parametro uguale a false significa obbligare Javascript a fermarsi finchè non riceve una

risposta da parte del server.

```
xmlHttp.open("GET", "data.php", true);
xmlHttp.onreadystatechange = …; // ne riparliamo più avanti
xmlHttp.send(); // invio della richiesta alla pagina
```

- **Metodo POST**:
  // Creazione dell'handler. La richiesta è di tipo POST ed è rivolta alla pagina *data.php* presente sullo stesso server del nostro documento. La richiesta DEVE essere asincrona.
  ```
  xmlHttp.open("POST","ajax_test.php",true);
  ```

  // Aggiunta di un intestazione dove indichiamo il tipo di codifica
  ```
  xmlhttp.setRequestHeader("Content-type","application/x-www-formurlencoded");
  ```

  ```
  xmlHttp.onreadystatechange = …; // ne riparliamo più avanti
  ```

  // Stessa funzione di prima, si indica nell'unico parametro i valori da inviare a *data.php*.
  ```
  xmlHttp.send("fname=Henry&lname=Ford");
  ```

3. **Scrivere una funzione `useHttpResponse`**. Stabilisco cosa deve fare Javascript quando il server risponde con successo alla mia richiesta.

```
xmlHttp.onreadystatechange = function() {
  // Sappiamo che la funzione viene eseguita quando cambia il valore di this.readyState
  // Con l'if stabilisco di muovermi solo se la richiesta ha avuto successo (quindi this.readyState == 4)
  // Devo anche verificare che la pagina non abbia restituito errore 403 o 404
  // (Vedere descrizioni delle proprietà nella tabella della pagina precedente)

  if (this.readyState == 4 && this.status == 200) {
    // Pongo la risposta in formato testo come valore di un input.
    document.mymodule.day.value = this.responseText;
  }
}
```

Funzione semplificativa (presa da w3schools)

- Se dobbiamo lavorare con AJAX può essere utile usare la seguente funzione: il primo argomento è l'URL della pagina con cui vogliamo interagire, il secondo è il nome della funzione da eseguire in caso di risposta positiva della pagina.

```
function loadDoc(url, cFunction) {
  var xhttp;
  xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      cFunction(this);
    }
  };
  xhttp.open("GET", url, true);
  xhttp.send();
}
```

- **Esempio**:

```
loadDoc("url-2", myFunction2);
function myFunction2(oggetto) {
  document.mymodule.day.value = oggetto.responseText;
}
```

La funzione `myFunction2` presenta un parametro: l'oggetto `XMLHttpRequest`.

Esempio di codice con uso di XML (da w3schools)

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      table,th,td {
        border : 1px solid black;
        border-collapse: collapse;
      }
      th,td {
        padding: 5px;
      }
    </style>
  </head>
  <body>
    <h1>The XMLHttpRequest Object</h1>

    <button type="button"
onclick="loadDoc()">Get my CD
collection</button>
    <br><br>
    <table id="demo"></table>

    <script>
    function loadDoc() {
      var xhttp = new XMLHttpRequest();
      xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
          myFunction(this);
        }
      };
      xhttp.open("GET", "cd_catalog.xml", true);
      xhttp.send();
    }

    function myFunction(xml) {
      var i;
      var xmlDoc = xml.responseXML;
      var table="<tr><th>Artist</th><th>Title</th></tr>";
      var x = xmlDoc.getElementsByTagName("CD");
      for (i = 0; i <x.length; i++) {
        table += "<tr><td>" + x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue
+ "</td><td>" + x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue + "</td></tr>";
      }
      document.getElementById("demo").innerHTML = table;
    }
    </script>
  </body>
</html>
```

```
▼<CATALOG>
 ▼<CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
 ▼<CD>
    <TITLE>Hide your heart</TITLE>
    <ARTIST>Bonnie Tyler</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
 ▼<CD>
    <TITLE>Greatest Hits</TITLE>
    <ARTIST>Dolly Parton</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>RCA</COMPANY>
    <PRICE>9.90</PRICE>
```

Esempio di file XML

Solito codice visto prima

Il file XML presenta una struttura molto simile a quella di un codice HTML. Le funzioni che utilizziamo per cercare elementi nel DOM possono essere usate per estrarre e gestire singolarmente gli elementi di un file XML.

Pagina risultante

The XMLHttpRequest Object

Get my CD collection

Artist	Title
Bob Dylan	Empire Burlesque
Bonnie Tyler	Hide your heart
Dolly Parton	Greatest Hits
Gary Moore	Still got the blues
Eros Ramazzotti	Eros
Bee Gees	One night only

Per un esempio con uso di JSON vedere i laboratori del prof.Tesconi