PHP:
INTRODUZIONE

Cos'è PHP

- E' un **linguaggio di scripting interpretato** utilizzato principalmente per la realizzazione di pagine web dinamiche (ma non solo!)
- Sintassi molto **simile al C** e al Perl
- Linguaggio a **tipizzazione debole**
- Linguaggio ad **alto livello**, con più di 3000 funzioni nelle API messe a disposizione di base
- Esistono **wrapper** per le tecnologie/librerie più utilizzate (MySqI, GD, OpenSSL, XML, FTP, CORBA ecc)

Sintassi essenziale (simile a *C*)

- **Tag di apertura/chiusura:**
  <?php e **?>** delimitano le porzioni di codice che devono essere interpretate dal parser PHP.

- **Variabili:** *si usa il carattere* **$** *davanti al nome della variabile stessa (che è case-sensitive). Inoltre non serve dichiarare le variabili.*

- **Tipizzazione:** sono supportati i classici tipi primitivi (es. **boolean**, **integer**, **float**, **string**, **array**, **object**), ed è inoltre previsto un tipo speciale **mixed** (accetta più tipi).
  tuttavia, il tipo di una variabile non è specificato dal programmatore, ma è **deciso a run-time** dall'interprete PHP in base al contesto.

- **Classi e oggetti:** sono supportate, ma differenti in PHP4/PHP5.

- Per il resto (costrutti di controllo e condizionali, terminazione di riga ecc) è del tutto analogo a C, Java e Javascript.

- Ampia gamma di **funzioni nella API**:
  - manipolazione di archivi *(zip, rar, bzip2)*
  - crittografia *(OpenSSL, Hash, Mcrypt)*
  - manipolazione del filesystem *(Direct IO, Directories, Fileinfo)*
  - image processing *(GD, ImageMagick, Exif)*
  - altri servizi *(Mail, FTP, PDF, SSH, LDAP)*

Principio di funzionamento

- PHP è utilizzato per **creare oggetti multimediali** *(pagine HTML, immagini, documenti XML, PDF ecc)*
- In pratica, **tutto quello che viene "scritto" tramite PHP** (es. con i comandi *echo* e *printf*) **viene restituito al browser** (nel nostro caso alla GUI)
- PHP deve comunicare al browser il **tipo del dato che sta generando** (con i cosiddetti "*headers HTTP*")
- Il tipo di default, se non specificato, è **text/html**

- Esempio:

```
<html>
    <head>
    <title>Pagina HTML di prova</title>
    </head>

    <body>
    Sono le <b><?php echo date('H:i'); ?></b> del giorno <b><?php
        echo date('d-m-Y');
    ?></b>
    </body>
</html>
```

Questo script genererà un documento di testo che verrà visualizzato dal browser come una semplice **pagina HTML** (tipo di default)

- Esempio:

```
<myXML>
   <info  date="<?php echo date('d-m-Y'); ?>"
          time="<?php echo date('H:i'); ?>"
   />
</myXML>
```

Questo script dovrebbe generare un documento di testo XML, ma sarà interpretato dal browser ancora come una semplice **pagina HTML**!

Si deve infatti informare il browser di interpretare il documento inviato nel modo appropriato (cioè XML).

- Esempio (continua):

```
<?php header("Content-type: text/xml"); ?>
<myXML>
   <info  date="<?php echo date('d-m-Y'); ?>"
          time="<?php echo date('H:i'); ?>"
   />
</myXML>
```

In questo modo il browser interpreterà come XML il documento creato dallo script PHP!

- Esempio:

```
<?php header("Content-type: image/jpeg"); ?>
.....
.....funzioni della libreria GD....
.....
```

Headers HTTP

- Possono essere considerati come dei “comandi” per **configurare i parametri del protocollo HTTP** (nello specifico sono stringhe)

- Esempi:
  “Content-type: text/xml”
  “Content-type: image/jpeg”
  “Content-Disposition: attachment; filename=’documento.pdf’”
  “Location: http://www.miosito.com/”
  “HTTP/1.0 404 Not Found”

- Il comando PHP per inviare headers HTTP è **header(“**…header string…**”);**

Note:

- Gli headers devono essere **inviati prima** di qualsiasi altro output! (anche spazi o linee vuote)

```
<myXML>
.....
<?php header("Content.type: text/xml"); ?>
</myXML>
```

- Spesso è preferibile **salvarsi il testo da restituire in una stringa** e "scriverla" sono se nessun errore si verifica:

```
<?php
$str = "<myXML>";
$str = $str . "</myXML>";
.....
header("Content.type: text/xml");
echo $XML;
?>
```

- Fare attenzione ad eventuali **messaggi di errore/warning** restituiti in automatico da PHP: andrebbero in conflitto con in documento stesso!

```
<myXML>
<?php header("Content.type: text/xml"); ?>
</myXML>
```

ERRORE: mancano gli apici per chiudere la stringa!

PHP, di default, stampa un messaggio di errore per informare il programmatore

- Per tenere conto di questo, si può **momentaneamente sopprimere l'output** per mezzo del *buffering*.

  Con **ob_start()** si inizia il buffering: da questo punto in poi l'output generato andrà in un buffer in attesa di:
  - ★ essere scartato con **ob_end_clean()**
  - ★ essere restituito normalmente al browser con **ob_end_flush()**

Passaggio di parametri

- Per passare dei parametri dalla GUI allo script PHP si può:
  - specificarli direttamente nella **querystring** dell'url del dataset in questione:

```
<dataset  src="http://sito.it/news.php?category=latest&page=1">
```

  - utilizzare il **metodo setQueryParam()** del dataset:

```
<handler name="onclick">
    myDataset.setQueryParam("category","latest");
    myDataset.setQueryParam("page","1");
    myDataset.doRequest();
</handler>
```

- utilizzare l'**oggetto lz.Param** assieme al metodo setQueryString() del dataset:

```
<button>Send
    <handler name="onclick">
        var p = new lz.Param();
        p.addValue( "category","latest", true );
        p.addValue( "page", "1" );
        myDATASET.setQueryString( p );
        myDATASET.doRequest();
    </handler>
</button>
```

NB: questo ultimo metodo permette di codificare il valore dei parametri per evitare problemi di codifica/interpretazione durante il trasferimento (*URI encoding*)

- La modalità di passaggio (**GET** o **POST**) è determinata dall'attributo *querytype*

- Lo script PHP riceve i parametri in **due array** **$_GET** e **$_POST**, a seconda della modalità di trasmissione utilizzata nella GUI:

```
<?php
    if (  isset( $_GET['category']  ) { ..... }
    if (  isset( $_GET['page']  ) { ..... }
?>
```

NB: per testare se i parametri sono stati passati, si possono, ad es, usare le funzioni isset() ed empty()

4° esercitazione

1. Implementare un semplice **meccanismo di autenticazione**:
    - supponendo un solo utente/password (no controllo in database);
    - visualizzi o meno la schermata principale dell'applicazione a seconda che l'utente/pasword siano corretti o meno.
2. Visualizzare in una griglia il **contenuto di una directory** sul server *(files, cartelle, data creazione, permessi ecc...)*
3. Inserire nella GUI dei campi di testo in un form ed effettuarne la **validazione con PHP** *(validazione server-side)*