

Informatica **U**manistica

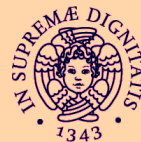
# Array, Funzioni e interazione con l'utente

Laboratorio di Progettazione Web

*AA 2009/2010*

*Chiara Renso*

*ISTI- CNR - [c.renso@isti.cnr.it](mailto:c.renso@isti.cnr.it)*



UNIVERSITÀ DI PISA

# Array

**Gli array sono contenitori in grado di memorizzare una sequenza di elementi indicizzati . Rispetto alla variabili possono contenere tanti valori diversi. Gli array di PHP sono di tipo associativo, ovvero associano un elemento ad una chiave**

Indice array o chiave

elements array

0	1	2	3	4
pippo	pluto	paperino	paperone	topolino

The diagram illustrates an array structure. It consists of a table with two rows and five columns. The top row contains the indices 0, 1, 2, 3, and 4. The bottom row contains the corresponding elements: pippo, pluto, paperino, paperone, and topolino. An arrow labeled 'Indice array o chiave' points to the top row, and another arrow labeled 'elements array' points to the bottom row.

# Creare un array

Gli array si creano con il costrutto array()

```
$myarray = array  
("pippo", "pluto", "paperino", "paperone", "topolino");
```

Abbiamo creato un array di cinque elementi, alla posizione 0 abbiamo "pippo", e così via fino alla quinta posizione (elemento 4).

# Creare un array

Gli array si possono creare anche assegnando i valori ai singoli elementi

```
$myarray[0]="pippo";
```

L'indice, se non specificato diversamente, è numerico ed inizia da 0. Altrimenti possiamo definire esplicitamente le chiavi sia numeriche che non.

```
$myarray1 = array(1=>"pippo",0=>"pluto",3=>"paperino",  
4=>"paperone",2=>"topolino");
```

```
$myarray2 = array  
( 'personaggio1'=>"pippo", 'personaggio2'=>"pluto", 'persona  
ggio3'=>"paperino", 'personaggio4'=>"paperone", 'personagg  
io5'=>"topolino")
```

# Accedere un array

L'accesso agli elementi può avvenire tramite l'indice

`$myarray[1]` ; indica l'elemento "pluto"

o tramite la chiave definita

`$myarray2['personaggio1'];`

# Accedere gli elementi di un array

Il costrutto foreach permette di effettuare cicli sugli elementi dell'array

**foreach(array as item)** dove item indica l'indice dell'array

Esempio:

```
foreach ($myarray as $item)
```

```
echo "$item <BR>";
```

# Aggiungere un elemento ad un array

`$array[] = $newelement;`

Viene aggiunto il nuovo elemento alla fine dell'array

`$myarray[]="archimede";`

# Array Multidimensionali

Un elemento di un array può contenere a sua volta un array, creando così un array multidimensionale

```
<? $myarray=array(array("pippo","pluto"),array  
("topolino","paperino"));
```

```
echo $myarray[0][1];
```

```
?>
```



# Manipolazione degli array

- ◆ Contare gli elementi: `count(array)` e `sizeof(array)`
- ◆ Randomizzazione di un array `shuffle(array)`
- ◆ `shuffle` richiede che il generatore di numeri casuale sia inizializzato dall'istruzione `srand`

# Randomizzare un array - esempio

```
<?
```

```
$pizze=array("margherita","capricciosa","quattro stagioni");
```

```
srand((double)microtime() * 1000000); // generatore random
```

```
shuffle($pizze);
```

```
echo $pizze[0];
```

```
?>
```

# Manipolazione di un array

Le funzioni **sort()** e **rsort()** permettono di ordinare un array in base agli elementi

**sort(\$myarray);** ordina in ordine crescente

**rsort(\$myarray);** ordina in ordine decrescente

Non restituiscono un valore ma ordinano direttamente l'array passato come parametro. Esempio

```
<?
```

```
$myarray=("uno","due","tre");
```

```
sort($myarray);
```

```
?>
```

# Manipolazione Array

**explode()** e **implode()** permettono di convertire un array in una stringa e una stringa in un array per mezzo di un carattere separatore.

In questo esempio `$mystring` è il risultato di `implode` dell'array dove gli elementi dell'array vengono separati nella stringa dal carattere spazio

```
$mystring = implode(" ", $myarray);
```

Analogamente, `$myarray` è il risultato di `explode` di una stringa dove gli elementi dell'array vengono distinti dal carattere barra |

```
$myarray = explode("|", $mystring);
```

# Funzioni

Le funzioni sono blocchi di codice che possono essere chiamati da più punti dello script php

```
function nome_funzione (arg_1,arg_2,...,arg_n)
```

```
{
```

```
.... Script....
```

```
return valore;
```

```
}
```

Le funzioni possono avere argomenti in input (*arg1,arg\_2, ... arg\_n*) e possono restituire un valore

Le variabili definite dentro una funzione hanno come ambito la funzione stessa. Quindi al di fuori della funzione non esistono.

# Funzioni

Creazione e chiamata di funzioni **con restituzione di valori**

```
<?php function add()
```

```
{    $somma=3+5;
```

```
return $somma;    }
```

```
.....
```

```
echo add();
```

```
?>
```

# Funzioni

Chiamata di funzione **senza restituzione di valori**

```
<? function stampa()
```

```
{
```

```
  echo "Funzione!";
```

```
}
```

```
stampa();
```

```
?>
```

# Funzioni con argomenti

```
<? function add1($a,$b)
```

```
{
```

```
$somma=$a+$b;
```

```
return $somma;
```

```
}
```

```
echo add1(3,5);
```

```
?>
```



# Funzioni

Le funzioni possono ritornare più valori, sotto forma di array

```
<? function tornaarray()  
  
{   $myarray=array("pippo","pluto","paperino");  
  
return $myarray;  
  
}  
  
.....  
  
$arraytornato=tornaarray();  
  
echo $arraytornato[1];  
  
?>
```

# Inclusione di file

Per condividere e riutilizzare porzioni di script e funzioni si usa frammentare il codice in più file in modo da poterlo includere quando serve. E' utile per modularizzare il codice in modo da non dover ripetere stessi pezzi di script in piu' parti (ad esempio connessione al database, header o footer di una pagina, form di login etc)

**include** (*nomefile*);

L'istruzione include va posizionata all'inizio dello script php o comunque prima dei riferimenti a variabili o funzioni definiti nel file da includere. L'interprete PHP segnala errore se il file non viene trovato, proseguendo comunque l'esecuzione

**require** (*nomefile*) è analogo tranne che genera un errore irreversibile qualora il file non venga trovato

# Interazione con l'utente

**Il grosso vantaggio dell'uso di tecnologie serverside sta nel poter gestire l'interazione con l'utente.**

**L'utente può interagire con una applicazione web (spedire parametri per la consultazione) principalmente con l'uso di Moduli o FORM.**

**Le Form sono costrutti di HTML e non di PHP. La parte di programma lato server permette di gestire i dati che vengono spediti al server tramite i costrutti delle Form.**

# Form

- ◆ L'HTML permette di visualizzare e formattare opportunamente le FORM, mentre la parte lato server riceve i dati spediti via HTTP e li gestisce.
- ◆ Il collegamento tra lato client (HTML) e lato server avviene nella action dove viene specificato lo script che riceve e gestisce i dati

## Parametri

Esempio di Form HTML

Nome:

Rosso  Verde

```
<FORM name=esempio action="pagina.php"  
METHOD=GET|POST>  
<INPUT type=text name=username>  
<INPUT type=radio name=color value="Rosso">  
...  
<INPUT type=submit>
```

# Metodi GET e POST

- ◆ Il passaggio dei parametri della Form può avvenire secondo due modalità: **GET** e **POST**.
- ◆ Il metodo GET prevede il passaggio dei parametri in una **QUERY STRING**, ovvero una stringa che racchiude i parametri **accodata alla URL**, quindi visibile dalla barra del browser. Lunghezza massima querystring 256 caratteri.
- ◆ Nel metodo POST i parametri vengono passati direttamente tramite protocollo HTTP, non sono visibili.
- ◆ Sono metodi equivalenti, ma il GET non viene usato in caso di password o in caso di molti parametri o passaggio di stringhe molto lunghe.
- ◆ Ogni metodo viene trattato in modo diverso dal lato server

# Metodo GET

Con il metodo GET i parametri vengono passati nella URL che riferisce la pagina chiamata dalla ACTION

Esempio:

```
http://localhost/submit.php?dato1=pippo&dato2=pluto
```

Il carattere **?** separa il nome della pagina dai parametri che vengono passati.

I parametri sono nella forma **nomeparametro=valore** separati dal simbolo **&**

Tutta la stringa dopo il simbolo **?** prende il nome di **querystring** o **stringa di interrogazione**

# Gestione dei parametri in PHP

PHP può accedere ai parametri in tre modi:

`$_POST[nomepar]` per il metodo POST

`$_GET[nomepar]` per il metodo GET

Oppure accedendo all'array globale delle richieste:

`$_REQUEST[nomepar]` per entrambi i metodi. In questo modo lo script PHP è indipendente dal metodo usato dalla FORM HTML

Tutti i parametri di passaggio nelle form sono nell'ambiente predefinito di php e sono quindi visualizzabili con la funzione `phpinfo()`

# FORM in PHP

## Radio button

- ◆ I valori inseriti come opzioni nei campi radio button si ottengono selezionando negli array `$_POST`/`$_GET` o `$_REQUEST` il nome del campo

In HTML:

Rosso `<INPUT type=radio name=mioradio value="rosso">`

Verde `<INPUT type=radio name=mioradio value="verde">`

In PHP:

```
$miopar1=$_REQUEST[mioradio];
```



# Form in PHP

- ◆ I campi checkbox, come gli altri, si accedono tramite gli array `$_GET` o `$_POST` o `$_REQUEST` selezionando il nome del campo della form

In HTML:

Rosso `<INPUT type=checkbox name=miocheckred value="rosso">`

Verde `<INPUT type=checkbox name=miocheckverde value="verde">`

In PHP:

```
$miopar2=$_REQUEST[miocheckred];
```

# Form in PHP

Analogamente, i campi di selezione si accedono con il nome

In HTML:

```
<select name=selezione>
```

```
<option> prima opzione</option>
```

```
<option> seconda opzione</option>
```

```
</select>
```

In PHP:

```
$miopar4=$_REQUEST[selezione];
```

```
echo $miopar4;
```

# Form in PHP

Nei campi di selezione multipli tramite il nome si accede ad un array che occorre indicare nel nome del campo select della form

In HTML:

```
<select name=selezione1[] multiple>
```

```
<option value="prima"> prima opzione</option>
```

```
<option value="seconda"> seconda opzione</option>
```

```
</select>
```

In PHP:

```
$miopar4=$_REQUEST[selezione1][0];
```

```
echo $miopar4;
```

# Form in PHP

Similmente per i campi textarea

```
<textarea name="miatext" rows=5>
```

```
</textarea>
```

In PHP:

```
$miopar5=$_REQUEST[miatext];
```

```
echo $miopar5;
```

# Pagine per form

- ◆ La gestione delle form prevede due passi: la visualizzazione della FORM in HTML e la gestione dei parametri in PHP. Possiamo immaginare di gestire questi passi in due pagine distinte, una in HTML solamente (quindi con estensione .html) e l'altra come pagina PHP (.php). In questo caso avremo, ad esempio, **miapaginaform.html** che descrive la form e che nella action indica la pagina **miapaginaform.php** per la gestione dei parametri.

# Gestione delle form

gestform.php

```
<?
$miavar=$_GET['nome'];
echo $miavar;
?>
```

gestform.html

```
<FORM name=miaform
action=gestform.php
method=get>
<INPUT type=text
name="nome">
....
<INPUT TYPE=SUBMIT>
```


← gestform.php?nome="pippo"

# Una sola pagina per gestire le form

- ◆ Talvolta questa modalità di gestione rischia di far proliferare eccessivamente il numero di pagine (html e php), si può quindi usare **una sola pagina PHP** sia per la visualizzazione della form che per la gestione dei parametri
- ◆ Per gestire sia la visualizzazione del form che la gestione nello stesso script occorre distinguere quando visualizzare la form e quando elaborare lo script.
- ◆ Occorrerà distinguere il caso di visualizzazione della parte della form dall'altra per l'elaborazione dei parametri.
- ◆ Questa distinzione verrà fatta in base al valore del campo di submit. Occorre quindi dare un nome al campo di submit e poi testarne il valore con `$_POST[submit]`

# Gestione Form

gestform.php



```
if (isset($_POST[submit])) {  
    // trattamento parametri  
    .....  
} else {  
    //visualizza la form  
    echo "<FORM  
    name=miaform  
    method=post  
    action=gestform.php >";  
    .....  
    <INPUT TYPE=submit  
    name=submit>}  
}
```



# Esempio:miaform.php

```
if (isset($_POST['submit'])) {
```

La pagina richiama se  
stessa come action

```
echo "trattamento parametri"; // trattamento parametri
```

```
} else { //visualizza la form
```

```
echo "<FORM name=miaform method=post  
action=miaform.php >";
```

```
echo "<INPUT name='miopar' type='text'>";
```

```
echo "<INPUT type='submit' name='submit'>";
```

```
echo "</FORM>"; }
```

Il pulsante di submit ha un  
nome tramite il quale  
testiamo se la pagina e'  
stata chiamata la prima  
volta o in seguito ad una  
sottomissione

## Esempio:miaform1.php

Si può richiamare la pagina stessa in modo parametrico usando la variabile predefinita `$_SERVER[PHP_SELF]`. E' un modo più pulito e scalabile, se si cambia nome allo script non occorre cambiare il codice!

```
<FORM name=miaform method=post action='<? echo  
$_SERVER[PHP_SELF] ?>' >;
```

```
<INPUT name=miopar type=text>;
```

```
<INPUT type=submit name="submit">;
```

```
</FORM>;
```

# Gestione delle form in PHP

- ◆ **La prima fase del trattamento dei parametri in genere prevede un controllo sulle correttezza dei dati.**
- ◆ **Questo controllo può essere fatto in parte in javascript per una interazione più diretta con l'utente ma, in parte o completamente, può essere fatta anche in PHP e quindi lato server.**
- ◆ **Possiamo quindi aggiungere alla nostra pagina di gestione delle form una funzione di controllo campi immessi**

# Controllo parametri

```
if (isset($_POST['submit'])) {  
  
    checkform(); // controllo parametri  
  
} else {    //visualizza la form  
  
    echo "<FORM name=miaform method=post action=miaform.php >";  
  
    echo "<INPUT name=nome type=text>";  
  
    echo "<INPUT name=cognome type=text>";  
  
    echo "<INPUT type=submit name='submit'>";  
  
    echo "</FORM>"; }  
}
```

# Controllo parametri

```
function checkform() {  
  
    if ($_POST['nome']== "") {  
  
        echo "inserire nome";  
  
    } else if ($_POST['cognome']== "") {  
  
        echo "inserire cognome";  
  
    }  
  
}
```