Gli Array

- Gli Array sono una delle funzioni più interessanti e flessibili.
- A differenza degli altri linguaggi.
  - Gli array di PHP possono archiviare dati di vari tipi.
  - Possono organizzarli automaticamente in vari modi.
- E' possibile elencare i modi comuni con cui vengono utilizzati
  - Molte variabili di ambiente sono array
  - La maggior parte delle funzioni dei database trasferiscono le loro informazioni tramite array
    - Creando un pacchetto compatto

Gli Array (2)

- Gli Array formano un contenitore per eseguire manipolazioni.
- Qualunque situazione che raccoglie pezzi di dati da **impacchettare** e **gestire** come cosa unica
  - E' adatta ad essere gestita con **Array**.
- Gli array di PHP sono array ***associativi***.
  - Con piccoli meccanismi supplementari.

Gli Array (3)

- **Associativi** significa che archiviano valori di elementi insieme a valori chiave.
  - Anziché in un rigoroso ordine lineare di indici.
- Gli array di altri linguaggi sono ***array vettori*** non ***associativi***
- Se si archivia un elemento in un array (insieme alla chiave)
  - Quello di cui si necessita per recuperarlo è il valore della chiave.

Gli Array (4)

Esempio:

```
<?
      $posizione['Perugia'] = 'Umbria';
      $regione = $posizione['Perugia'];
      echo $regione;
?>
```

Output:

Umbria

Gli Array (5)

- Se si vuole unire un **ordine numerico** ad un gruppo di valori
  - Bisogna utilizzare numeri interi come valori chiave

Esempio:

```
$myarray[1] = “il mio primo dato”;
$myarray[2] = “il mio secondo dato”;
```


Gli Array (6)

- Gli Array possono essere **multidimensionali**.
  - Possono archiviare valori insieme a sequenze di chiavi
  - Anziché una singola chiave.
- Gli array mantengono una ***lista ordinata*** degli elementi inseriti
  - Indipendentemente da quelli che sono i valori chiave.
- Ciò permette di trattare gli array come **liste**!

Vettori vs. Associativi

- Gli Array vettori sono a ***dichiarazione*** e ***tipi*** **anticipati**
  - Sono molto veloci
  - Sia per l’archiviazione che per la valutazione
- Il compilatore alloca l’array in un **blocco di memoria contiguo!**
- Gli array associativi *(alcuni li chiamano hash)*
  - Anziché avere un numero fisso di slot
  - Vengono aggiunti ***quando serve***
- Invece di pretendere che gli elementi siano dello stesso tipo
  - Gli array in PHP hanno la stessa **flessibilità** delle variabili.

Vettori vs. Associativi (2)

- Gli **Array vettori** allocano i loro elementi in **ordine numerico**.
  - Le chiavi utilizzate devono avere ***valori interi***.
- Gli array di PHP possono avere chiavi di tipo ***arbitrario***.
  - Comprese chiavi di tipo ***STRING.***
  - Si possono avere assegnazioni consecutive del tipo:

```
$myarray[1]=1;
$myarray[“rosso”] = 2;
$myarray[3] = 3;
```

Array con indici numerici

- Se non specificato diversamente, gli indici sono numerici

```
$Capitali= array("Parigi", "Vienna", "NewYork");

0 -> "Parigi"
1 -> "Vienna"
2 -> "New York"
```


Array con indici numerici (2)

- L' accesso agli elementi in un array indicizzato si effettua con:
  - **nome** dell'array
  - **indice** dell'elemento

Esempio:

```
$Capitali[0]  ->  "Parigi"
```


Gli Array: Crearli

- Ci sono 3 modi per creare un array in uno script PHP:
  - **Assegnano** un valore all'interno dell'array
  - **Utilizzando** la funzione `array()`
  - **Chiamando** una funzione che restituisca un array.

Gli Array: Assegnazione Diretta

- E' il modo più semplice!
- Comportarsi come se una variabile ***fosse*** già un array.
- Basterà assegnargli un valore:

```
$myarray[1]="Il mio primo array";
```

- Se 'myarray' era una variabile ***non array***
  - Diventerà un array con un elemento
- Se 'myarray era ***un array***
  - L'elemento sarà ***aggiunto*** in coda insieme alla chiave
  - Se '1' è già utilizzato il vecchio valore verrà ***sostituito***.

Gli Array: La funzione `array()`

- **`array()`** crea un nuovo array attraverso
  - La descrizione dei suoi elementi e delle chiavi associate.
- Un elenco di elementi da archiviare **separati** da virgole.
  - Il sistema penserà ad archiviarli in maniera sequenziale.
  - Utilizzando delle **chiavi numeriche** incrementali.

```
$carrello=array('frutta','salumi','caffè');
```

- L'array **ricorderà** l'ordine in cui sono stati archiviati gli elementi!

Gli Array: La funzione `array()`

```
$carrello=array('frutta','salumi','caffè');
```

- E' equivalente a scrivere:

```
$carrello[0] = 'frutta';
$carrello[1] = 'salumi';
$carrello[2] = 'caffè';
```

Se $carrello non era esistente, prima:

```
$carrello[] = 'frutta';
$carrello[] = 'salumi';
$carrello[] = 'caffè';
```


Gli Array: La funzione `array()`

- array() offre una sintassi speciale per specificare
  - ***qualunque*** tipo di indice per l'array
- Invece dei valori degli elementi separati da virgole
  - Si forniscono **coppie** chiave-valore separate da virgole.
  - Chiave e valore sono separati da **=>**

```
$carrello=array(0=>'frutta',1=>'salumi',2=>'caffè');
```


Gli Array: La funzione `array()`

Esempio:

```
$a=array('rosso'=>'mela','arancio'=>'arancia',
         'giallo'=>'banana');
echo $a['giallo'];
```

Output:

banana

Gli Array: array restituiti da funzioni

- Chiamando una funzione che restituisce array.
  - Può essere una funzione definita dall'utente
  - Può essere una funzione built-in
  - Molte funzioni di interazione con DB restituiscono array.
- Altre funzioni creano array per altre funzioni.
  - Una di queste è **range()**
  - Utilizza due interi come argomenti e restituisce un array
  - Con tutti gli interi che ci sono tra gli argomenti

Gli Array: array restituiti da funzioni

Esempio:

```
$myarray = range(1, 5);
```

Equivale a:

```
$myarray = array(1, 2, 3, 4, 5);
```


Gli Array: Recupero di valori

- Il modo più diretto per recuperare un valore è utilizzando l'indice.
  - Se **`$myarray[5]`** ha un valore, esso verrà restituito.
  - Altrimenti si comporterà come una variabile indefinita.

Esempio:

```
<? $a=array('rosso'=>'mela','arancio'=>'arancia');
echo $a['giallo'];
$a[5]="valore strano";
echo $a[5];
echo $a[8]; // questo non esiste!
?>
```


Gli Array: La costruzione list()

- Viene sfruttato il fatto che gli array registrono in maniera silenziosa l'ordine con il quale sono stati archiviati gli elementi.
- **list()** assegna diversi elementi di array a variabili in successione.

```
$carrello=array(0=>'frutta',1=>'salumi',2=>'caffè');
list($prima_busta, $seconda_busta) = $carrello;
```

- Darà come risultato l'assegnazione di frutta alla prima busta
- I salumi alla seconda busta
- Il caffè lo portiamo a mano (non abbiamo abbastanza buste!)

Gli Array: La costruzione list()

- Le variabili verranno assegnate nell'ordine di registrazione degli elementi nell'array.
- list() è ***l'opposto*** (in comportamento) di array
  - array() **impacchetta** il suo argomento in un array
  - list() **divide** l'array in assegnazioni individuali di variabili

Array multidimensionali

- È possibile creare array con più di una dimensione, semplicemente definendo ogni elemento come un array.

Esempio:

```
$punti = array( array(1,2),
                array(5,8),
                array(3,1));
```

- In realtà è anche possibile definire ogni elemento del primo array separatamente
  - non necessariamente come array.

Array multidimensionali

- Pensare agli array come multidimensionali rende le cose più difficili di quanto non lo siano.
- Basta ricordare che i valori archiviati negli array possono essere essi stessi array!
- Si possono avere diverse profondità di riferimento in diverse parti dell'array:

```
$marray[0]="una stringa";
$marray[1]['contiene']="una stringa più profonda";
```


Gli Array: Cancellazione

- Cancellare un elemento da un array è semplice
- Come eliminare una variabile assegnata
- Basta chiamare **`unset()`**

```
$myarray[0]='voluto';
$myarray[1]='NON voluto';
$myarray[2]='voluto';
unset($myarray[1]);
```

- Le chiavi restano inalterate!

Gli Array: Iterazione

- Gli array creano un elenco ordinato coppie chiave/valore
  - Nello stesso ordine in cui sono stati inseriti
- Il motivo è supportare operazioni che si **ripetono**
  - Non è facile farlo con un **for()**
  - Potrebbero esserci indici ***non numerici***
- Negli array è come se ci fosse un **puntatore nascosto!**
  - Ogni coppia ***punta*** alla seguente
  - Quando si aggiunge il primo elemento **il puntatore** indica il primo vero elemento

Gli Array: Iterazione

```
$info=array();
$info[0]='Caracas';
$info['Caracas']='Venezuela';
$info[1]='Parigi'; $info['Parigi']='Francia';
$info[2]='Tokio'; $info['Tokio']='Giappone';

function citta_per_numero($indice, $array_) {
     if(IsSet($array_[$indice])) {
          $a = $array_[$indice];
          $b = $array_[$a];
          echo "la città " . $a . "è in: " . $b;
     }
}
```


Gli Array: Iterazione

- Se vogliamo stampare tutto ciò che è presente nell'array, senza conoscerne la struttura precisa?
- Si può stampare il contenuto con le funzioni di iterazione
- **current()** e **next()**

```
function stampa_tutto($array_) {
     $tmp=current($array_);
     if($tmp) echo "$tmp<BR>";
     else echo "nulla da stampare";
     while($tmp = next($array_) echo "$tmp<BR>";
}
```


Gli Array: Iterazione

- La funzione current restituisce il valore archiviato alla posizione attuale del nostro ***puntatore***
- Sarà il primo elemento se l'array viene creato ***fresh***
- se next viene chiamata sull'ultimo valore, restituisce ***falso***
- Ricordarsi che le chiamate a funzione di PHP sono ***"chiamate per valore"***
- Ogni funzione riceverà una nuova copia degli argomenti

Operare su Array

0	Roma
1	Milano
2	Catania
3	Firenze
4	Parigi
5	Bologna

```
current($citta)="Firenze";
```


Operare su Array

0	Roma
1	Milano
2	Catania
3	Firenze
4	Parigi
5	Bologna

```
next($citta);
current($citta)="Parigi";
```


Operare su Array

0	Roma
1	Milano
2	Catania
3	Firenze
4	Parigi
5	Bologna

```
previous($citta);
current($citta)="Firenze";
```


Gli Array: Reset()

- Per tornare all'inizio dell'array (spostare il puntatore)
- Si utilizza reset()
- si può sostituire la chiamata a current() con reset()

```
function stampa_tutto($array_) {
      $tmp=reset($array_);
      if($tmp) echo "$tmp<BR>";
      else echo "nulla da stampare";
      while($tmp = next($array_) echo "$tmp<BR>";
}
```

Gli Array: Ordine Inverso

- Esiste anche **prev()** ed **end()**
  - **prev()** *(sposta il puntatore indietro di una posizione)*
  - **end()** *(sposta il puntatore all'ultimo elemento)*

```
function stampa_tutto($array_) {
      $tmp=end($array_);
      if($tmp) echo "$tmp<BR>";
      else echo "nulla da stampare";
      while($tmp = prev($array_) echo "$tmp<BR>";
}
```


Gli Array: Estrazione di Chiavi

- E' possibile recuperare il valore delle chiavi.
  - **key()** si comporta come **current()**.
  - Restituisce la chiave anziché il valore.

```
function stampa_tutto($array_) {
      $tmp=reset($array_);
      $tmp2=key($array_);
      if($tmp) echo $tmp2 . ":" . "$tmp<BR>";
      else echo "nulla da stampare";
      while($tmp = prev($array_) {
            $tmp2 = key($array_);
            echo $tmp2 . ":" . "$tmp<BR>";
      }
}
```


Gli Array: Estrazione di Chiavi

- Tutti gli esempi hanno un BUG!
  - se il contenuto è ***false***, il ciclo si interrompe
- Esiste **each()**
  - E' simile a **next()**
  - Restituisce ***false*** solo quando termina l'array.
  - Ritorna non solo i valori, ma coppie di chiave/valore in un nuovo array!

Gli Array: Estrazione di Chiavi

Per accedere a tutti gli elementi di un array si può utilizzare il costrutto **`for`**, unitamente all'istruzione **`each`**.

`each` prende la chiave (o l'indice) e il valore dell'elemento corrente dell'array e li mette a sua volta in un nuovo array di 2 elementi

- key: indice
- value: valore

Inoltre, sposta il cursore avanti di una posizione.

Gli Array: Estrazione di Chiavi

```
function stampa_tutto($array_) {
      $tmp=reset($array_);
      while($tmp = each($array_)) {
            $a = $tmp['value'];
            $b = $tmp['key'];
            echo $b . ":" . "$tmp<BR>";
      }
}
```

- Non è stato fatto un controllo sul primo elemento
- **Next()** restituisce dopo che incrementa (perdiamo un elemento)
- **Each()** restituisce il valore **prima** di spostare in avanti il **ptr**

Ciclo for con gli array

- Per accedere in sequenza a tutti gli elementi di un ciclo è sufficiente chiamare **each** sull'array un numero di volte uguale al numero di elementi dell'array:

```
for ($i = 0; $i <= $N; $i++)
{ $Elemento = each ($Array);
  echo "chiave: $Elemento[key]";
  echo "valore: $Elemento[value]";
}
```

- La funzione **count($Nome_Array)** restituisce la lunghezza dell'array.

Ciclo for con gli array

Esempio

$Num è la lunghezza dell'array che contiene gli ingredienti (**$Ingr**)

Il ciclo che segue stampa di tutti gli elementi dell'array degli ingredienti come celle di una tabella

```
for ($i = 0; $i < $Num; $i++) {
  $Riga = each($ingr);
  $Num = $Riga[key] + 1;
  echo "<tr><td>ingrediente $Num :</td>
  <td>$Riga[value] </td></tr>";
}
```


Iterazioni negli Array associativi

- Gli elementi degli array associativi possiedono comunque un indice oltre alla chiave numerica.
- In un array associativo gli indici vengono assegnati in base all'ordine di inserimento.
- Si possono utilizzare le tecniche per l'iterazione in un array indicizzato anche con gli array associativi.

Ordinamento di un Array

- Gli elementi di un array indicizzato numericamente sono ordinati grazie agli indici.
- Se si omettono gli indici, l'ordine di inserimento determina l'ordinamento dell'array.
- È possibile ordinare alfabeticamente un array sulla base dei suoi contenuti con la funzione `sort($Mio_Array).`

Ordinamento di un Array

- Se si ordina un array associativo con **sort()**
  - Le chiavi inserite dall'utente vengono sostituite da indici numerici.
- Per ordinare alfabeticamente un array associativo a seconda dei suoi contenuti senza distruggere le chiavi si utilizza la funzione
- `asort($Mio_Array).`

Conversione di stringhe in array

`implode()` converte un array in una stringa.

Ha due argomenti: il separatore, ad es. "," l'array che deve essere convertito in stringa.

```
$Stringa = implode(", ",$carrello)
echo $Stringa;
```

stampa: frutta, salumi, caffè

`explode()` converte una stringa in un array.

Anche `explode` ha un argomento separatore, che delimita un elemento dall'altro (ad esempio, uno spazio).

Funzioni per gli array

`is_array ()` restituisce vero se l'argomento è un array.

`count() size_of()` restituisce la dimensione dell'array.

`in_array()` restituisce vero se il primo argomento è presente nel secondo (l'array).

`unset($mio_array[1])` cancella l'elemento dato dall'array.

Array e WHILE

Si può utilizzare il anche costrutto while insieme alle istruzioni `each` e `list` nella condizione:

```
while(list($Indice,$Valore)=each($Mio_Array))
{ echo "$Indice - $Valore";
}
```

`list` assegna alle variabili `$Indice` e `$Valore` rispettivamente l'indice e il valore dell'elemento restituito da `each`;

Quando si è scorso tutto l'array, each restituisce un valore nullo, rendendo così falsa la condizione.

Array nelle Stringhe

- All'interno di una stringa è possibile accedere al valore degli elementi di un array.

```
echo "valore inserito: $Mio_Array[1]";
echo "capitale della Francia: $Capitali[Francia]";
```

- Notare che all'interno di una stringa la chiave non è compresa tra virgolette doppie!