PHP: Le basi del linguaggio

Introduzione

- Nato come linguaggio dinamico per scrivere pagine web personali
  - Nato come **P**ersonal **H**ome **P**age
  - Ora: **P**HP: **H**ypertext **P**reprocessor
- È alla versione 5 e contempla una serie grandissima di librerie per
  - gestione stringhe
  - accesso a vari database (MySQL, PostGres, etc)
  - XML
  - cookies
  - creazione dinamica di immagini
  - …
- Open source, per ogni piattaforma
- Versione descritta in queste slides: 5.x

PHP: documentazione

- Sito ufficiale: **http://www.php.net**
  - sorgenti, documentazione online anche in italiano
- Altri siti utili:
  - **http://www.phpbuilder.com** (community con articoli specifici e forum)
  - **http://www.hotscripts.com** (scripting in generale, esempi e programmi pronti)

PHP: caratteristiche principali

- Linguaggio interpretato
- Sintassi di derivazione C
- Utilizzabile sia in modo procedurale tradizionale che object-oriented (con limitazioni)
- Linguaggo «loosely typed»
  - Non è necessario dichiarare le variabili
  - Conversione automatica tra tipi quando necessario

Incorporare PHP nelle pagine HTML

- I files PHP debbono essere inseriti nella parte di file system visibile al server web, insieme alle normali pagine HTML
- Nel caso di XAMPP, le pagine sono da porsi nella cartella htdocs
- Solitamente le pagine HTML che incorporano codice PHP hanno estensione .php
  - Web servers che supportano il PHP passano queste pagine automaticamente all'interprete PHP
  - Comunque, comportamento configurabile (es. passare tutti i .html per non mostrare che php è usato)

Incorporare PHP nelle pagine HTML

- Le pagine HTML pure non vengono modificate dall'interprete PHP
- Il codice PHP può essere incorporato nelle pagine in differenti modi:
  - Dentro a una «php preprocessing instruction»: **<?php … ?> (RACCOMANDATO!)** o tramite tag di «general processing» (se configurati in php.ini): **<? … ?> <% … %>**
  - All'interno di HTML script elements:
    <script language="php">……</script>

Iniziare con il PHP

- Un metodo semplice per verificare che l'interprete PHP sia configurato e operi correttamente è chiamare la funzione **`phpinfo()`**
  - Accedere tramite un browser al server web all'URL che corrisponde ad una pagina il cui contenuto è:

```
<?php
phpinfo();
?>
```

Sintassi di base

- Fine statement: `;` (punto e virgola) – obbligatorio, si possono anche spezzare gli statement su più linee.
- Commenti:
  monolinea: `//commento`
  `#altro commento`
  multilinea `/* righe di commenti … */`

- Le keywords, come le funzioni e i metodi, sono case **IN**sensitive
- Il resto è case sensitive (es. variabili)

Tipi di dati

- Tipi scalari:
  - bool o boolean (valori TRUE e FALSE)
  - Integer (dipendente dal sistema, tipicamente 32-bit, sintassi C per le costanti)
  - float (dipendente dalla piattaforma, tipicamente doppia precisione, sintassi C per le costanti)
  - string
- Tipi compositi:
  - array (indicizzati e associativi)
  - oggetti
- Per verificare il tipo si usano le funzioni

  `is_int()`, `is_float()`, `is_bool()`, `is_string()`

Stringhe

- Le stringhe possono essere scritte usando varie notazioni (possono coprire più linee):
  - Singoli apici
    - Non viene fatta l'espansione delle sequenze di eacape e delle variabili, tranne \' per l'apice singolo e \\ per il backslash
    - Esempio: `'Sample string with \' and \\ '`
  - Doppi apici
    - Espansione delle sequenze di escape e delle variabili
    - Esempio:

```
$color = 'white';
echo "the horse\nis $color"
```

Stampa:

```
the horse
is white
```

Costanti simboliche

- È possibile associare dei nomi a delle costanti

```
define('Nomecostante', Valore)
```

```
define("color","rosso");
define('prezzo',13.50);
echo color; //stampa rosso
```

Magic constants

- Ci sono delle costanti simboliche predefinite, che iniziano per doppio underscore (_ _)

`__FILE__`	the full name of the current file
`__DIR__`	the directory of the current file
`__LINE__`	the current line number
`__FUNCTION__`	the current function name
`__CLASS__`	the current class name
`__METHOD__`	the current method name
`__NAMESPACE__`	the current namespace

Operatori utili

- Operazioni aritmetiche: `+ - * / ++ --`
  - (non c'è la divisione intera! (cioè con troncamento), il risultato e' float a meno che il resto sia zero)
- Operazioni logiche: `==, ===, !=, !==, !`, NOT, `&&` , AND, `||` , OR, `.....`
- Concatenazione di stringhe: `.` (punto)
  - es. `$s = "Lato "."server" ;`
  - abbreviato: `$s .= "stringa"` equivale a `$s=$s."stringa"`
- Prestare attenzione alla precedenza degli operatori

Variabili

- In PHP, le variabili sono prefissate da **$** (segno di dollaro)
- Le variabili iniziano con una lettera o _ e contengono solo lettere, cifre o underscore (_)
- Sono case sensitive
- Le variabili possono essere locali o globali
- Esempi:

```
$a = "prova";
$b = 18;
$c = $b+77;
```

Esempi

- L'operatore di assegnamento è l'uguale

```
$A = 1;
$B = "2";
$C = ($A + $B); // Somma di interi
$D = $A.$B; // Concatenazione di strighe
echo $C; // Stampa 3
echo $D; // Stampa "12"
$4F = "ciao"; //Errore nel nome della var
```

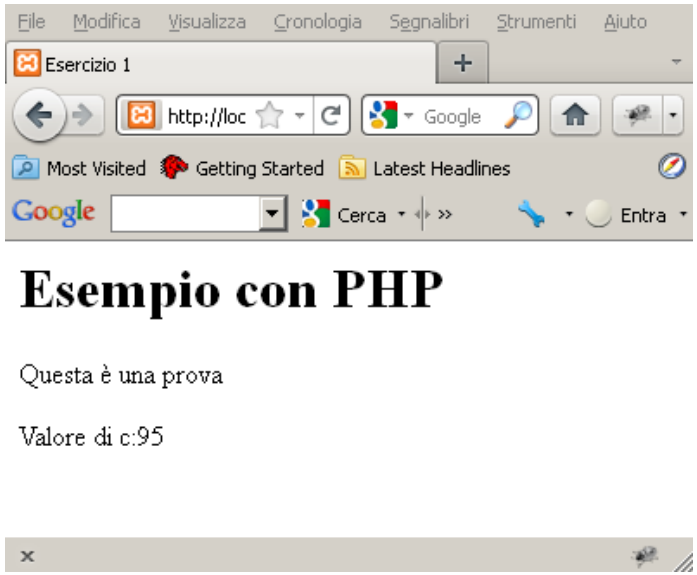
Primo programma

```
<html>
      <head>
  <title>Esercizio 1</title></head>
  <body>
      <h1>Esempio con PHP</h1>
      <p>
      <?php
            $a = "prova";
            $b = 18;
            $c = $b+77;
            echo "Questa &egrave; una $a
    </p>";
            echo "<p>Valore di c:$c </p>";
       ?>
  </body>
</html>
```

example1.php

Risultato

```
<html>
        <head>
   <title>Esercizio 1</title></head>
   <body>
        <h1>Esempio con PHP</h1>
        <p>
        <?php
                $a = "prova";
                $b = 18;
                $c = $b+77;
                echo "Questa &egrave; una $a </p>";
                echo "<p>Valore di c:$c </p>";
        ?>
   </body>
</html>
```



Ancora sulle variabili

- La function **isset()** controlla che la variabile argomento sia definita (true) o meno (false)
- Si può fare riferimento ad una variabile in modo indiretto

```
$prova='uno';
$$prova=3;//assegna 3 a $uno
$rosso=& $giallo /*entrambe le var. fanno
   rif. alla stessa area di memoria */
```

In realtà una variabile è semplicemente un riferimento tramite un nome ad una zona della memoria

Distruzione delle variabili

- Di solito non serve
- Si può usare **unset()**

```
$rosso=& $giallo;
unset($giallo);   // non distruggo $rosso
```

Altri operatori

- @
  - Quando precede uno statement, gli eventuali errori generati non vengono riportati (nel log)
- ` (backtick)
  - La stringa all'interno viene passata allo shell ed il risultato è quello dell'esecuzione del comando

```
$output=`dir`;
echo $output;
```

Cast esplicito

- PHP ha un meccanismo di casting implicito ma supporta anche quello esplicito (sintassi tipo C)
- Esempio:

```
<?php
$a = 56;
$b = 12;
$c = $a/$b;  // $c è 4.66666 (float)
$d = (int)($a/$b);  // $d è 4 (int)
?>
```

Statement if

- ```
  if (condizione)
        istruzione;
  ```
- ```
  if (condizione)
        istruzione1;
    else
        istruzione2;
  ```
- ```
  if (condizione1)
        istruzione1;
    elseif (condizione2)
        istruzione2;
    …
    else
        istruzione3;
  ```
  ```

Cicli

- **while (condizione)**
  istruzione;
- **do**
  istruzione
  while(condizione);

- **for(inzial.;condizione;incremento)**
  istruzione;

Switch

```
switch($x)
  {
  case value1: istruzione1;
           break;
  case value2: istruzione2;
            break;
  …
  default: istruzione3;
}
```

I valori possono essere numeri o stringhe, il confronto è fatto con == (loosely typed)

Sintassi alternativa

- La { di un blocco è sostituita da :
- La } di un blocco è sostituita da:

endif; endwhile; endfor; endforeach; endswitch;

- Esempio:

```
<?php
if ($a == 5):
  echo "a equals 5";
  echo "...";
else:
  echo «a is not 5";
endif;
?>
```

Mix di controllo e HTML

- Il codice PHP (anche singoli statements) possono essere spalmati su più processing instructions con codice HTML nel mezzo.
- Esempio:

```
<?php if ($expression == true): ?>
This will show if the expression is true.
<?php else: ?>
Othersize this will show.
<?php endif; ?>
```

Mix di controllo e HTML

- Il codice PHP (anche singoli statements) possono essere spalmati su più processing instructions con codice HTML nel mezzo.
- Esempio:

```
<?php if ($expression == true): ?>
This will show if the expression is true.
<?php else: ?>
Othersize this will show.
<?php endif; ?>
```

Funzioni definite dall'utente

- Formato tipo sintassi *C*

```
function nome( lista param. )
{ istruzioni }
```

- La lista parametri è opzionale ed usa come separatore la virgola
- Il nome della funzione è case insensitive
- La lista dei parametri sono variabili (che iniziano per $)
- Esempio:

```
function sumsquares($a,$b)
{ return ($a*$a+$b*$b); }
```

Chiamate di funzione

- Formato tipo sintassi C

```
nome( lista param. )
```

- Esempio:

```
echo sumsquares(2,3);  // prints 13
```

- Quando chiamata, la funzione deve essere già stata processata dall'interprete PHP
- Si possono chiamare molte funzioni predefinite
- Si possono fare chiamate con numero variabile di parametri

Numero variabile di argomenti

- Per accedere ai parametro di una funzione (oltre ai parametri formali) vi sono le funzioni:
  - `$a=func_get_args()//array di arg.`
  - `$b=func_get_num()//num. di arg.`
  - `$val=func_get_arg[$i]//valore arg. $i`
- Il risultato di queste funzioni non può essere usato direttamente in espressioni, deve essere assegnato a delle variabili

```
function foo()
{ $numargs = func_num_args();
echo "Number of arguments: $numargs\n"; }
```

Regole di scoping

- Variabili definite dentro la funzione
  - Local scope : dentro la funzione nella quale è definita (gli argomenti della funzione rientrano in tale categoria)
- Variabili definite al di fuori delle funzioni
  - Scope: tutto il codice al di fuori delle funzioni (non visibili dentro le funzioni)
- Variabili globali dentro le funzioni
  - Una variabile dichiarata dentro una funzione con il qualificatore `global` ha scope globale
- Variabili statiche
  - Definite dentro una funzione tramite `static` sono come variabili locali ma mantengono il valore quando la funzione termina (come in C)

Esempio

```
$acc = 0;
function sum($x) {
      global $acc;
      $acc += $x;
}
sum(10); sum(10);
echo $acc;   // prints 20
```

scoping.php

Variabili «superglobal»

- Variabili globali predefinite
  - Sono visibili e accessibili dovunque
  - Hanno la forma di array associativi
  - Tipicamente usate per informazioni di ambiente
- Esempi:
  - **$GLOBALS** tutte le variabili globali correntemente definite nello script
  - **$_GET** tutte le variabili passate allo script via HTTP GET
  - **$_POST** tutte le variabili passate allo script via HTTP POST

  - **$name = $_GET['name'];**

Passaggio parametri

- I parametri possono essere passati per valore o indirizzo
- Il default è il passaggio per valore
- I parametri passati per indirizzo sono preceduti da **&**
- Esempio:

```
Function add_some_extra(&$string) {
    $string .= 'and something extra.';
}
$str = 'A string, ';
add_some_extra($str);
echo $str; // prints 'A string, and something extra.'
```

Risultato «by reference»

- Il risultato (return) può essere passato per reference o per valore (default = valore)
  - Se il nome della funzione è preceduto da &, la funzione ritorna un referece
- Esempio:

```
function &get_x_ref() {
    static $x = 30;
    return $x;
}
$y = &get_x_ref(); // now $y is an alias for $x
echo $y; // prints 30
```

Argomenti con valore di default

- Un valore di default può essere specificato per ogni argomento (come un assegnamento)
- Esempio:

```
function conc($a, $b, $sep=', '){
return ($a.$sep.$b);
}
echo conc("First", "Second", ',');
echo conc("First", "Second");

// entrambi stampano la stessa cosa
```

Mettere funzioni in una variabile

- Una variabile che contiene il nome di una funzione può essere usata per invocare la funzione stessa
- Esempio:

```
function average($a,$b,$c){
return ($a+$b+$c)/3;
}
$av="average";
echo $av(1,2,3); // prints 2
```

Funzioni anonime

- Una funzione anonima può essere creata chiamando la funzione **create_function**
- Esempio:

```
$lambda=create_function('$a,$b,$c',
      'return ($a+$b+$c)/3');
echo $lambda(1,2,3); // prints 2

// parameters
// body
```

Terminazione dello script

- Le funzioni **`exit()` , `die()`** terminano l'esecuzione dello script
  - Possono prendere una stringa o un intero come parametri
  - La stringa è stampata prima della terminazione
  - L'intero è ritornato
- Esempio:

```
exit("connection failed");
```

Eccezioni

- Modello simile a quello di altri linguaggi (es. Java)
- Statement **`throw`** per segnalare una eccezione
- Statement **`try . . . catch`** per individuare e trattare le eccezioni

Esempio

classe standard

```
<?php
   function nfatt($x) {
    if (!is_int($x)) {
        throw new Exception('Operando non intero.\n');}
     elseif ($x<0){
        throw new Exception('Operando negativo.\n')}
     elseif ($x>13){
        throw new Exception('Operando troppo grande'.\n')}
     else {
        for($s=1;$x>0;$x--)
                $s*=$x;
    return $s;}
   }//fine function nfatt
.....................................................
   try {
       echo nfatt(5) . "\n";
       echo nfatt(3.5) . "\n";
   } catch (Exception $e) {
       echo 'Eccezione: ',  $e->getMessage();
   }
```

Gestione implicita delle eccezioni

- Si può definire una funzione di trattamento errori non trattati da statement **catch**
- Si usa la seguente function

set_exception_handler("*nome funct.*")

- Se viene eseguito un throw senza catch, viene chiamata la funzione indicata nel param.
- Alla fine dell'esecuzione dell'handler, l'esecuzione termina

Esempio

```
<?php
  function exception_handler($exception) {
   echo "Eccezione non catturata: ",
            $exception->getMessage(), "\n";
  }// fine handler
……
  set_exception_handler('exception_handler');
……
  throw new Exception('Eccezione non prevista');

  echo "Parte non eseguita\n";
  ?>
```

Arrays

- I vettori in PHP possono contenere dati di tipo eterogeneo
  - I vettori sono indirizzabili sia con un indice numerico che con un'etichetta di testo (array associativo)
- Creazione:
  - esplicita: `$vettore=array(7,1967,"carciofi",12);`
  - implicita: `$vettore[0]=7; $vettore[1]=1967; …`
  - associativa: `$t['nome']="Barney"; $t['cogn']="Gamble";`

  `$t=array('nome'=>"Barney",'cogn'=>'Gamble');`
- Accesso ai componenti:
  - `$vettore[$i]=… ;$tabella['nome']='Homer'; …`

Esempio 2

```
<?php
$passwords=array(
   'bart' => 'calzino',
   'homer' => 'birra',
   'lisa' => 'nobel',
   'marge' => 'caspiterina',
   'maggie' => '' );
$numprimi[0] = 1;$numprimi[1] = 2;$numprimi[2] = 3;
?>
<html>
      <head><title>Esempio 2</title></head>
  <body>
      <h1>Array</h1>
      <?php
      $indice=2; $nome="homer";
      echo "<p>Password di $nome:
                  $passwords[$nome]</p>";
      echo "<p>Il primo in posizione $index:
            $numprimi[$indice]</p>";
      ?>
  </body>
</html>
```

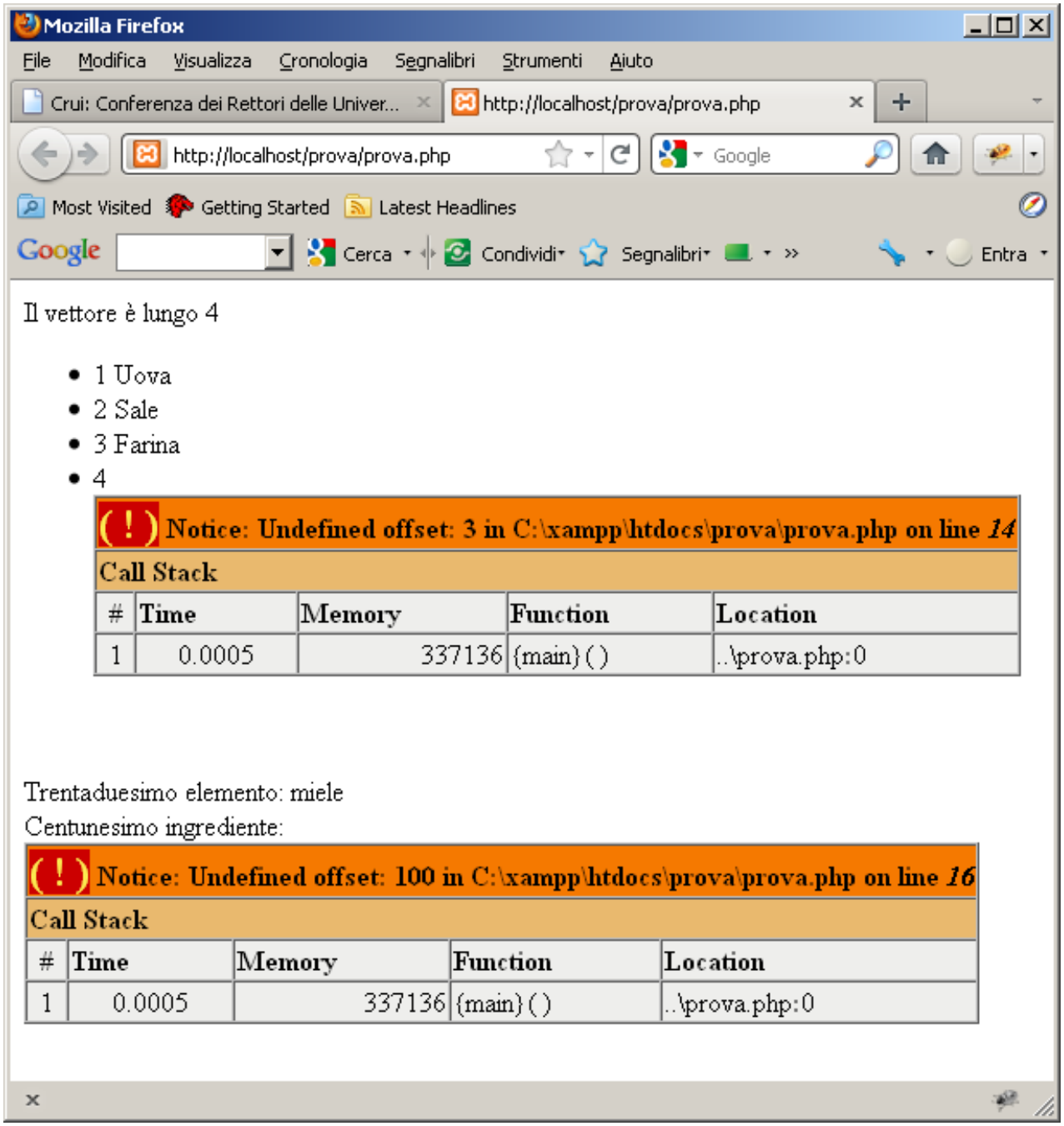
example2.php

Esempio

```
<?php
$ingredienti[] = "uova";
$ingredienti[] = "sale";
$ingredienti[2] = "farina";
$ingredienti[31] = "miele";
echo "<P> Il vettore &egrave; lungo ",
   count($ingredienti),"</P>";
echo "<UL> \n " ;
$c=count($ingredienti);
for($i=0; $i< $c; $i++) {
echo "<LI> ", $i+1;
echo " ",ucwords($ingredienti[$i]), " <br>"; }
// ucwords = uppercase , $incredienti[3] ERRORE

echo "</UL> <BR> Trentaduesimo elemento:
   ",$ingredienti[31],"<br>";
echo "Centunesimo ingrediente: ", $ingredienti[100];
?>
```

example3.php


Mozilla Firefox
File Modifica Visualizza Cronologia Segnalibri Strumenti Aiuto
Crui: Conferenza dei Rettori delle Univer...
http://localhost/prova/prova.php
http://localhost/prova/prova.php
Google
Most Visited Getting Started Latest Headlines
Google Cerca Condividi Segnalibri Entra
Il vettore è lungo 4
1 Uova
2 Sale
3 Farina
4
Notice: Undefined offset: 3 in C:\xampp\htdocs\prova\prova.php on line 14
Call Stack
Time Memory Function Location
1 0.0005 337136 {main}( ) ..\prova.php:0
Trentaduesimo elemento: miele
Centunesimo ingrediente:
Notice: Undefined offset: 100 in C:\xampp\htdocs\prova\prova.php on line 16
Call Stack
Time Memory Function Location
1 0.0005 337136 {main}( ) ..\prova.php:0

Accesso alle chiavi

- Si può non conoscere i valori degli indici in un array associativo
- Si può usare la function **each($nome_array)**
- Esempio

```
$a=array('nome'=>"Mario",'cogn'=>'Rossi',
  'nato'=>"13-4-1978");
$c=count($a);
for($i=0;$i<$c; $i++)
{$elem=each($a);
echo $elem['key']," ",$elem["value"], "
  <br>\n";
}
```

Altre funzioni per iterazione

- **`current()`**
  - Restituisce il valore dell'elemento corrente
- **`key()`**
  - Restituisce il valore della chiave dell'elemento corrente
- **`reset(), end()`**
  - Sposta il puntatore al primo/ultimo elemento
- **`next(), prev()`**
  - Sposta il puntatore al prossimo/precedente elemento

Cicli foreach

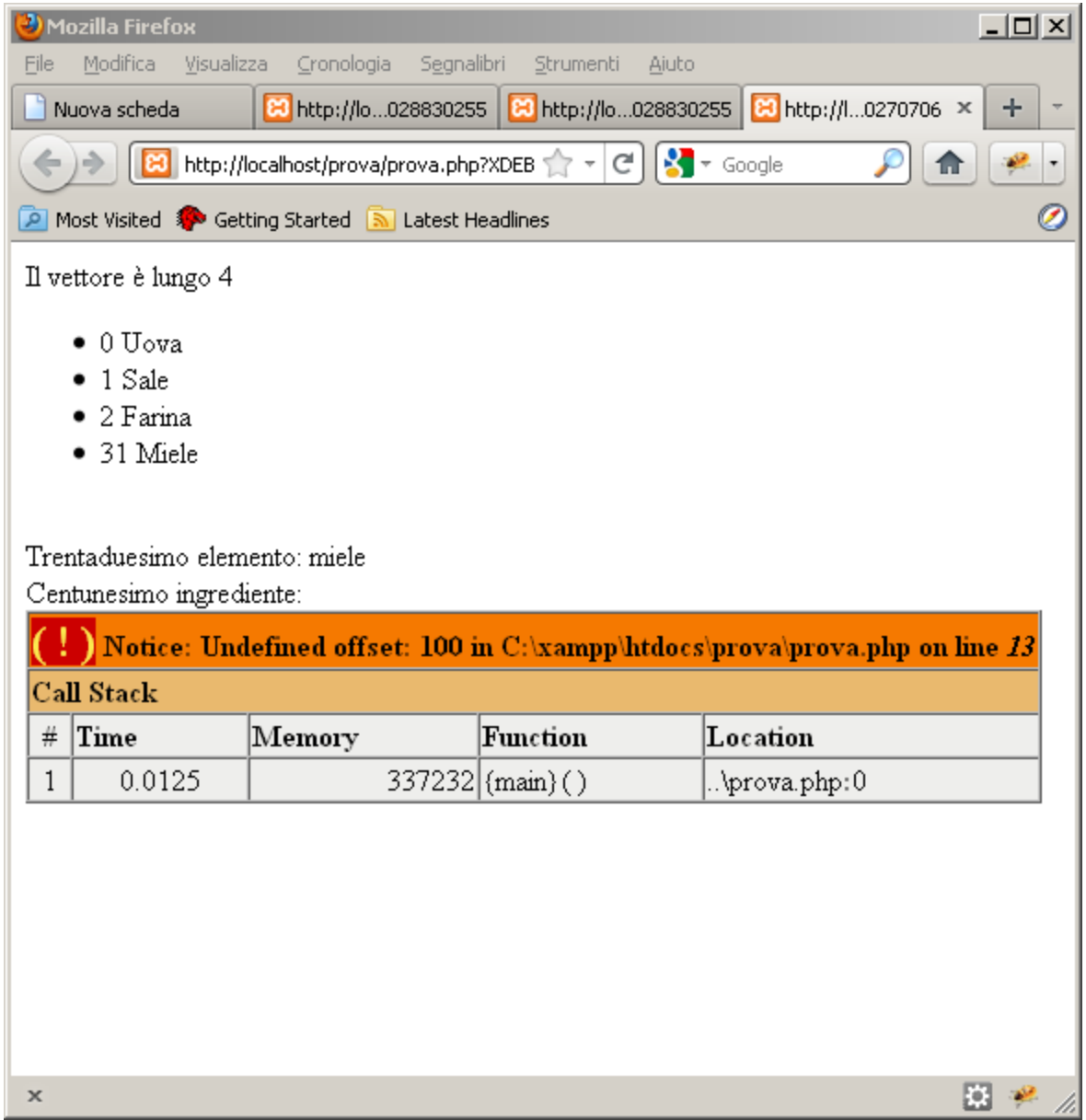
- Permettono di iterare sugli elementi di un array
- `foreach($nome_array as $contatore)`
  `istruzioni;`
- `$contatore` assumerà tutti i valori contenuti nell'array
- `foreach($vettore as $chiave=>$valore)`
  `istruzioni;`
- `$chiave $valore` esploreranno tutte le chiavi e valori nell'array

Stampa ingredienti senza (quasi) errori

```
<?php
$ingredienti[] = "uova";
$ingredienti[] = "sale";
$ingredienti[2] = "farina";
$ingredienti[31] = "miele";
echo "<P> Il vettore &egrave; lungo ",
   count($ingredienti),"</P>";
echo "<UL> \n " ;
$c=count($ingredienti);
foreach ($ingredienti as $chiave=>$valore){
echo "<LI> ", $chiave;
echo " ",ucwords($valore)," <br>"; }
echo "</UL> <BR> Trentaduesimo elemento:
   ",$ingredienti[31],"<br>";
echo "Centunesimo ingrediente: ", $ingredienti[100];
?>
```

example4.php

Risultato


Mozilla Firefox
File Modifica Visualizza Cronologia Segnalibri Strumenti Aiuto
Nuova scheda
http://lo...028830255
http://lo...028830255
http://l...0270706
http://localhost/prova/prova.php?XDEB
Google
Most Visited
Getting Started
Latest Headlines
Il vettore è lungo 4
0 Uova
1 Sale
2 Farina
31 Miele
Trentaduesimo elemento: miele
Centunesimo ingrediente:
Notice: Undefined offset: 100 in C:\xampp\htdocs\prova\prova.php on line 13
Call Stack
Time Memory Function Location
1 0.0125 337232 {main}( ) ..\prova.php:0

Principali funzioni per gli array

- **`is_array()`** ritorna true se il parametro è un array
- **`count()`** ritorna il numero di elementi nell'array
- **`sort()`** ordina l'array. Altri parametri opzionali specificano come ordinare.
- **`explode()`, `compact()`** creano un array da: stringhe (spezzettandole) o nomi di variabili (compact)
- **`extract()`** crea variabili da un array