# Arrays

## What is an Array?

- An array is an ordered collection of elements. Each element has a value, and is identified by a key. Each array has its own unique keys.
- The keys can be either integer numbers or strings.

## The `array()` Construct

- Calling The `array()` construct creates a new array. Passing a series of values to the `array()` construct will populate the new created array with these values.
- Each one of the values will automatically get an index number, that will be its key.
- We can alternatively specify both the keys and the values.

## The `array()` Construct

```
<?php

$vec_1 = array(2,4,5);
echo "<BR><BR>"."simple array of numbers";
for($i=0; $i<3; $i++)
{
echo "<BR>".$vec_1[$i];
}

?>
```

## The `array()` Construct

```
<?php

$vec_1 = array("moshe","david","john");
echo "simple array of strings";
for($i=0; $i<3; $i++)
{
echo "<BR>".$vec_1[$i];
}

?>
```

## The `array()` Construct

```
<?php

$vec_1 = array(100=>"moshe",101=>"david",102=>"john");
echo "simple array of strings and their keys";
echo "<BR>".$vec_1[100];
echo "<BR>".$vec_1[101];
echo "<BR>".$vec_1[102];
?>
```

## The `array()` Construct

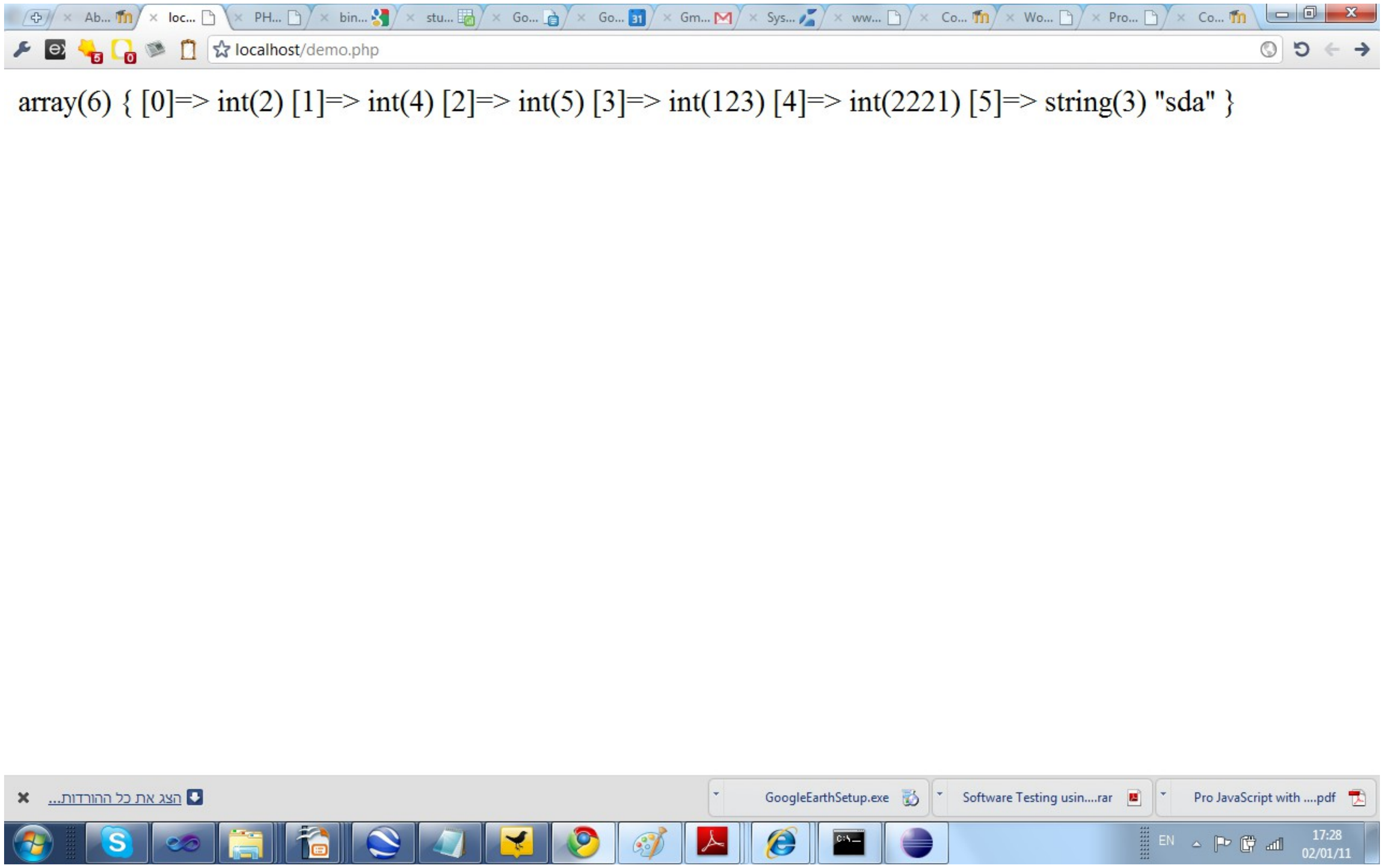
```
<?php

$vec_1 = array("m"=>"moshe","d"=>"david","j"=>"john");
echo "simple array of strings and their keys";
echo "<BR>".$vec_1["m"];
echo "<BR>".$vec_1["d"];
echo "<BR>".$vec_1["j"];
?>
```

# The `var_dump()` Function

- Prints out the content of a composite value (e.g. array) together with the data type of each one of the values.

```
<?php
$vec = array(2,4,5,123,2221,”sda”);
var_dump($vec);
?>
```

# The `var_dump()` Function



```
array(6) { [0]=> int(2) [1]=> int(4) [2]=> int(5) [3]=> int(123) [4]=> int(2221) [5]=> string(3) "sda" }
```

## The `var_dump()` Function

- Using `var_dump()` function we can print out more than one array.

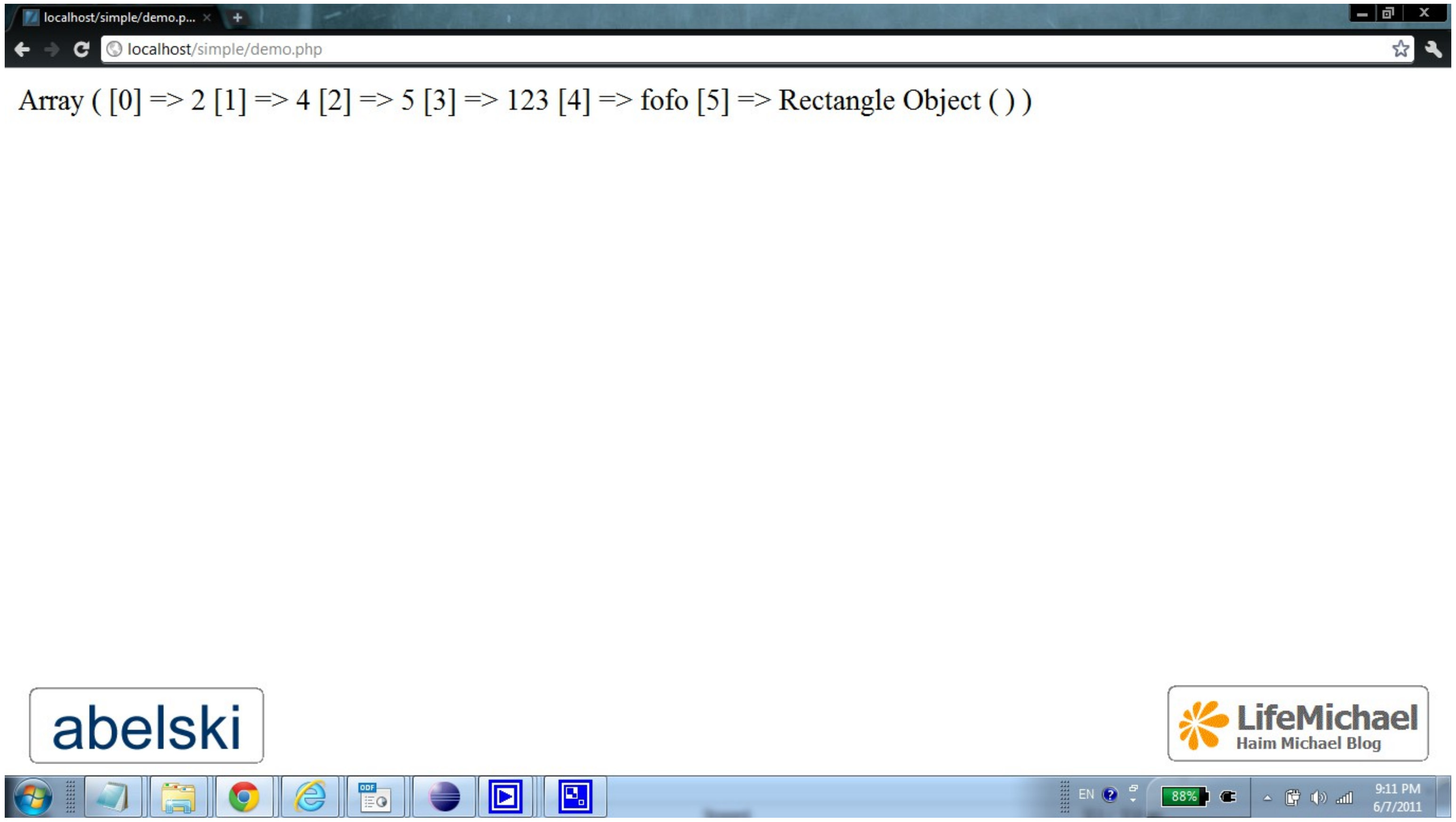
```
<?php
$vec_1 = array(2,4,5,123,2221);
$vec_2 = array(24,442,32,84,110);
$vec_3 = array(10,20,30,40,50);
var_dump($vec_1,$vec_2,$vec_3);
?>
```

## The `print_r()` Function

- Prints out the contents of a composite value (e.g. array).
- Unlike `var_dump()`, this function cannot print out more than one array.

```
<?php
class Rectangle {}
$vec_1 = array(2,4,5,123,”fofo”,new Rectangle());
print_r($vec_1);
?>
```

# The `print_r()` Function



```
Array ( [0] => 2 [1] => 4 [2] => 5 [3] => 123 [4] => fofo [5] => Rectangle Object ( ) )
```

## Array Inner Structure

- PHP arrays behave like ordered map. As such, they allow various possibilities:

  PHP arrays can be used to simulate different types of structures (e.g. map, queue, stack etc...).

  PHP arrays can have unique keys, both numeric and textual. When using numeric ones, they don't need to be sequential.

## Multi Dimensional Arrays

- A multidimensional array is an array that each one of its elements is another array.

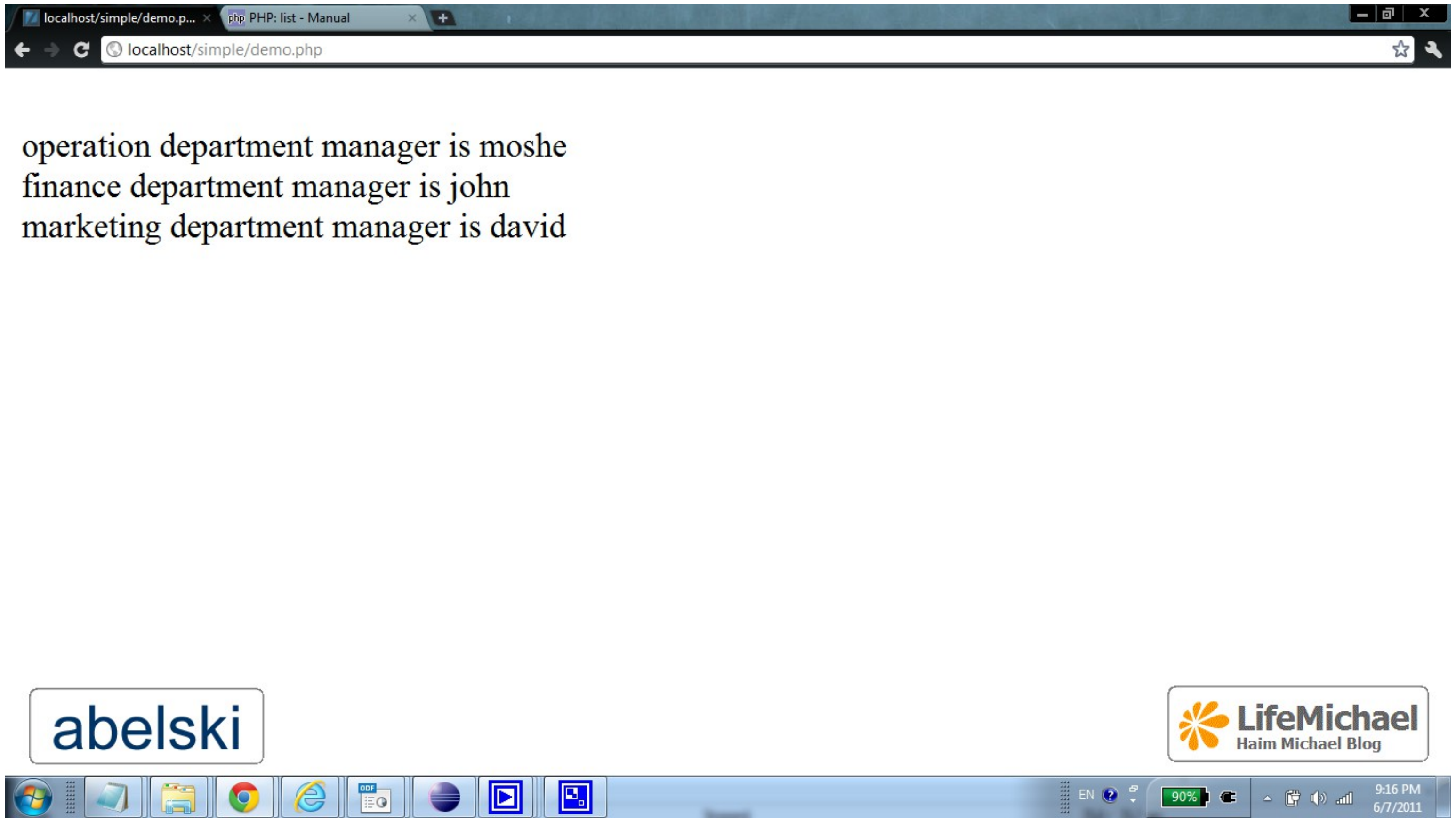
```
<?php
$matrix = array();
$matrix[0] = array("a","b");
$matrix[1] = array("c","d");
echo $matrix[0][0];
echo $matrix[0][1];
echo $matrix[1][0];
echo $matrix[1][1];
?>
```

## The `list()` Construct

- The `list()` construct provides a short cut for an automatic assignment of an array's elements into individual variables.

```
<?php
$info = array('moshe', 'david', 'michael','john');
list($operation_manager, $marketing_manager, ,$finance_manager) = $info;
echo "<BR>operation department manager is $operation_manager";
echo "<BR>finance department manager is $finance_manager";
echo "<BR>marketing department manager is $marketing_manager";
?>
```

# The `list()` Construct

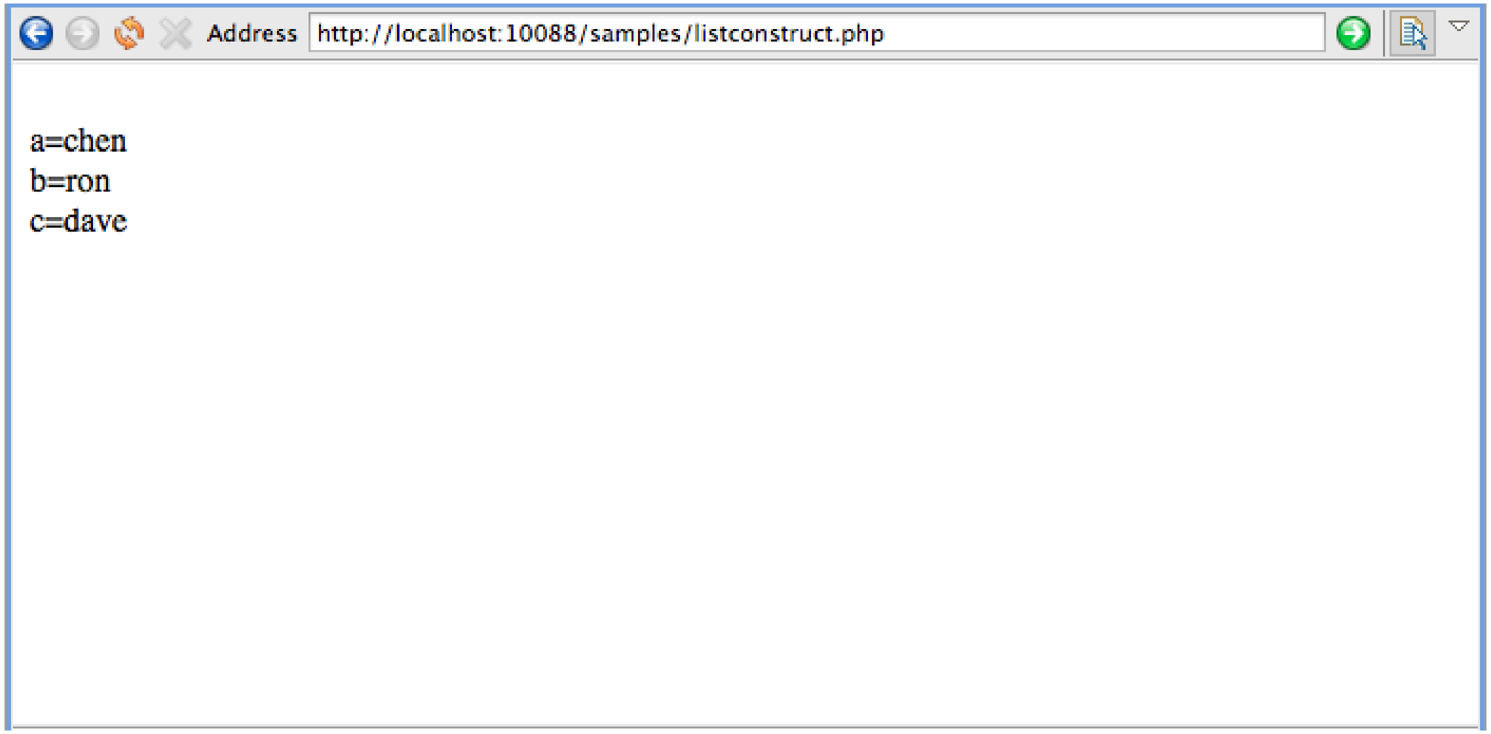


```
operation department manager is moshe
finance department manager is john
marketing department manager is david
```

# The `list()` Construct

```
<?php
$vec = [2=>"dave",1=>"ron",0=>"chen",3=>"ran"];
list($a,$b,$c) = $vec;
echo "<br>a=$a";
echo "<br>b=$b";
echo "<br>c=$c";
?>
```

## The `list()` Construct



```
http://localhost:10088/samples/listconstruct.php
```

```
a=chen
b=ron
c=dave
```

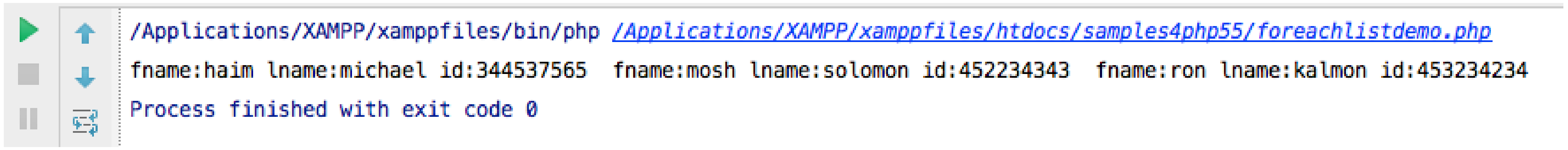
## The `list()` Construct

- As of PHP 5.5 we can use the list construct together with the foreach loop.

## The `list()` Construct

```php
<?php
$matrix = [
    ["haim", "michael", 344537565],
    ["mosh", "solomon", 452234343],
    ["ron","kalmon",453234234]
];

foreach ($matrix as list($fname, $lname,$id))
{
    echo "fname:$fname lname:$lname id:$id  ";
}
?>
```

# The `list()` Construct



```
/Applications/XAMPP/xamppfiles/bin/php /Applications/XAMPP/xamppfiles/htdocs/samples4php55/foreachlistdemo.php
fname:haim lname:michael id:344537565  fname:mosh lname:solomon id:452234343  fname:ron lname:kalmon id:453234234
Process finished with exit code 0
```

The Output

## The '+' Operator

- Using the + operator on two arrays we will get a union of the two arrays.
- Union of two arrays will include a union of the keys each one of the two arrays have and the values assigned with each one of them.

# The '+' Operator

```
<?php
$vec_1 = array(1,2,3);
$vec_2 = array(3,4,5,6);
$vec_3 = $vec_1 + $vec_2;
var_dump($vec_3);
?>
```

The Output

```
array(4) { [0]=> int(1) [1]=> int(2) [2]=> int(3) [3]=> int(6) }
```

## The '==' and '===' Operators

- The '==' operator (equality) returns true if the following condition fulfills:

  1. The two arrays contain the same elements.

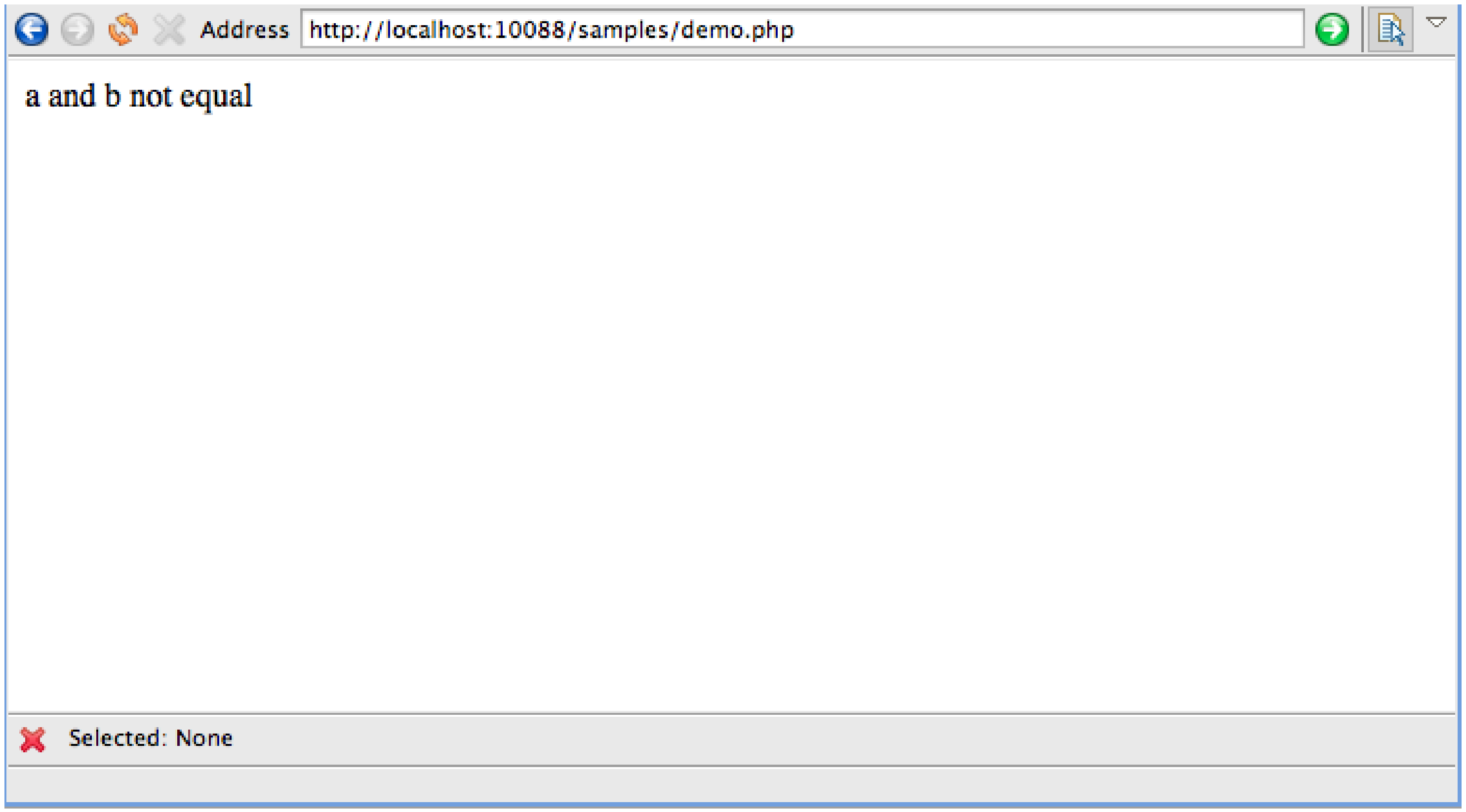
- The '===' operator (non identity) returns true if each one of the following two conditions fulfills:

  1. The two arrays contain the same elements.
  2. The two arrays have their identical elements in the same position.

## The '==' and '===' Operators

```
<?php
//$a = ['a'=>'avodado','b'=>'bamba','c'=>'calco'];
//$b = ['b'=>'bamba','a'=>'avodado','c'=>'calco'];
$a = [123,455,323];
$b = [323,123,455];
if($a==$b)
{
    echo "a and b equal";
}
else
{
    echo "a and b not equal";
}
?>
```

# The '==' and '===' Operators



```
a and b not equal
```

## The '!=' and '!==' Operators

- The '!=' operator (inequality) returns true if the following condition doesn't fulfill:

  1. The two arrays contain the same elements.

- The '!==' operator (non identity) returns true if at least one of the following two conditions doesn't fulfill:

  1. The two arrays contain the same elements.
  2. The two arrays have their identical elements in the same position.

## The '==', '===', '!=' and '!==' Sample

```
<?php
$vec_1 = array(1,2,3);
$vec_2 = array(1,2,3);
$vec_3 = array(0=>1, 1=>2, 2=>3);
$vec_4 = array(12=>1, 3=>2, 4=>3);
$vec_5 = array(1=>2, 0=>1, 2=>3);
var_dump($vec_1==$vec_2);     //true
var_dump($vec_1===$vec_2);    //true
var_dump($vec_1!=$vec_2);     //false
var_dump($vec_1!==$vec_2);    //false
echo "<BR>";
var_dump($vec_2==$vec_3);     //true
var_dump($vec_2===$vec_3);    //true
var_dump($vec_2!=$vec_3);     //false
var_dump($vec_2!==$vec_3);    //false
?>
```

## The '==', '===', '!=' and '!==' Sample

```
<?php
echo "<BR>";
var_dump($vec_2==$vec_4);     //false
var_dump($vec_2===$vec_4);    //false
var_dump($vec_2!=$vec_4);     //true
var_dump($vec_2!==$vec_4);    //true
echo "<BR>";
var_dump($vec_2==$vec_5);     //true
var_dump($vec_2===$vec_5);    //false
var_dump($vec_2!=$vec_5);     //false
var_dump($vec_2!==$vec_5);    //true
?>
```

## The `count()` Function

- Calling the `count()` function on a given array returns its size.

```
<?php
$vec = array(1,2,3,4,5,6,7,8);
echo count($vec);
?>
```

## The `is_array()` Function

❖ Calling the `is_array()` function on a variable returns true if that variable holds an array, and false if isn't.

```
<?php
$vec_1 = array(1,2,3,4,5,6,7,8);
$vec_2 = 123;
if(is_array($vec_1))
    echo "<BR>vec_1 is an array";
else
    echo "<BR>vec_1 is not an array";
if(is_array($vec_2))
    echo "<BR>vec_2 is an array";
else
    echo "<BR>vec_2 is not an array";
?>
```

## The `isset()` Function

- Calling the `isset()` function can tell us if a specific key already exists in our array... or not.

```
<?php
$vec = array('a'=>1,'b'=>2,'c'=>3);
if(isset($vec['a'])) echo "<BR>'a' key exists";
if(isset($vec['b'])) echo "<BR>'b' key exists";
if(isset($vec['c'])) echo "<BR>'c' key exists";
if(isset($vec['d'])) echo "<BR>'d' key exists";
if(isset($vec['e'])) echo "<BR>'e' key exists";
?>
```

## The `array_key_exists()` Function

- Calling the `array_key_exists()` function can tell us if a specific key already exists in our array... or not.

```
<?php
$vec = array('a'=>1,'b'=>2,'c'=>3);
if(array_key_exists('a',$vec)) echo "<BR>'a' key exists";
if(array_key_exists('b',$vec)) echo "<BR>'b' key exists";
if(array_key_exists('c',$vec)) echo "<BR>'c' key exists";
if(array_key_exists('d',$vec)) echo "<BR>'d' key exists";
if(array_key_exists('e',$vec)) echo "<BR>'e' key exists";
?>
```

## The `array_key_exists()` Function

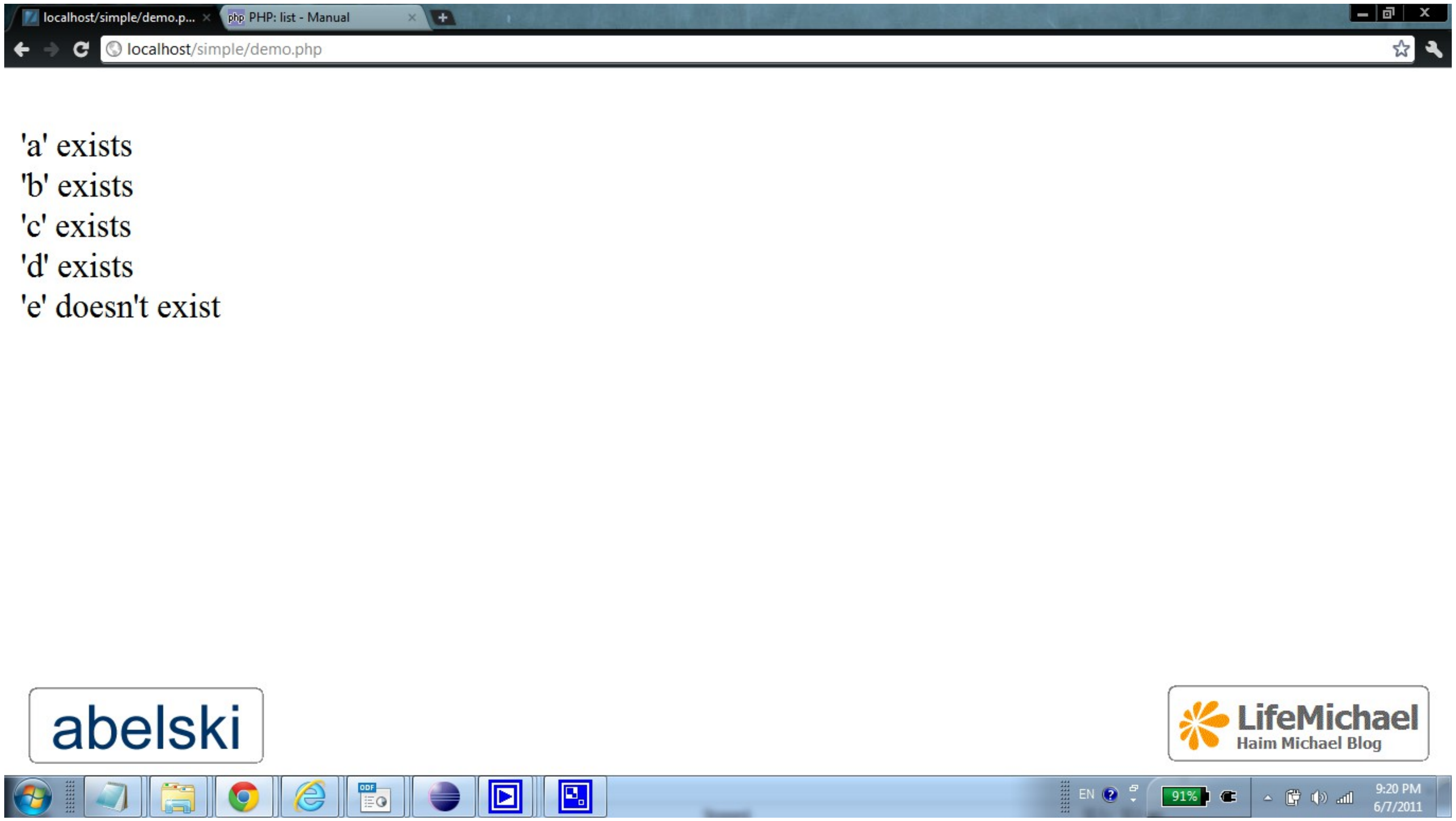
- The `isset()` function doesn't return `true` for array keys that were set together with `null` as its value.
- The `array_key_exists()` function does return `true` in those cases.

## The `in_array()` Function

- Calling the `in_array()` function checks if a given value exists in a given array.

```
<?php
$vec = array('a','b','c','d','f','g','h');
if(in_array('a',$vec)) echo "<BR>'a' exists";
else echo "<BR>'a' doesn't exist";
if(in_array('b',$vec)) echo "<BR>'b' exists";
else echo "<BR>'b' doesn't exist";
if(in_array('c',$vec)) echo "<BR>'c' exists";
else echo "<BR>'c' doesn't exist";
if(in_array('d',$vec)) echo "<BR>'d' exists";
else echo "<BR>'d' doesn't exist";
if(in_array('e',$vec)) echo "<BR>'e' exists";
else echo "<BR>'e' doesn't exist";
?>
```

# The `in_array()` Function



```
'a' exists
'b' exists
'c' exists
'd' exists
'e' doesn't exist
```

## The `array_flip()` Function

❖ This function returns a new array, which is the result of inverting value of each element with its key.

```
<?php
$vec_1 = array("a","b","c","d","f","g","h");
echo "<BR>before...<BR>";
var_dump($vec_1);
$vec_2 = array_flip($vec_1);
echo "<BR>after...<BR>";
var_dump($vec_2);
?>
```

# The `array_reverse()` Function

- This function returns a new array, which is the result of reversing the order of a given one.

```
<?php
$vec_1 = array("a","b","c","d","f","g","h");
echo "<BR>before...<BR>";
var_dump($vec_1);
$vec_2 = array_reverse($vec_1);
echo "<BR>after...<BR>";
var_dump($vec_2);
?>
```

## The Array Pointer

- When going over the elements, there is a pointer that points at the current element.

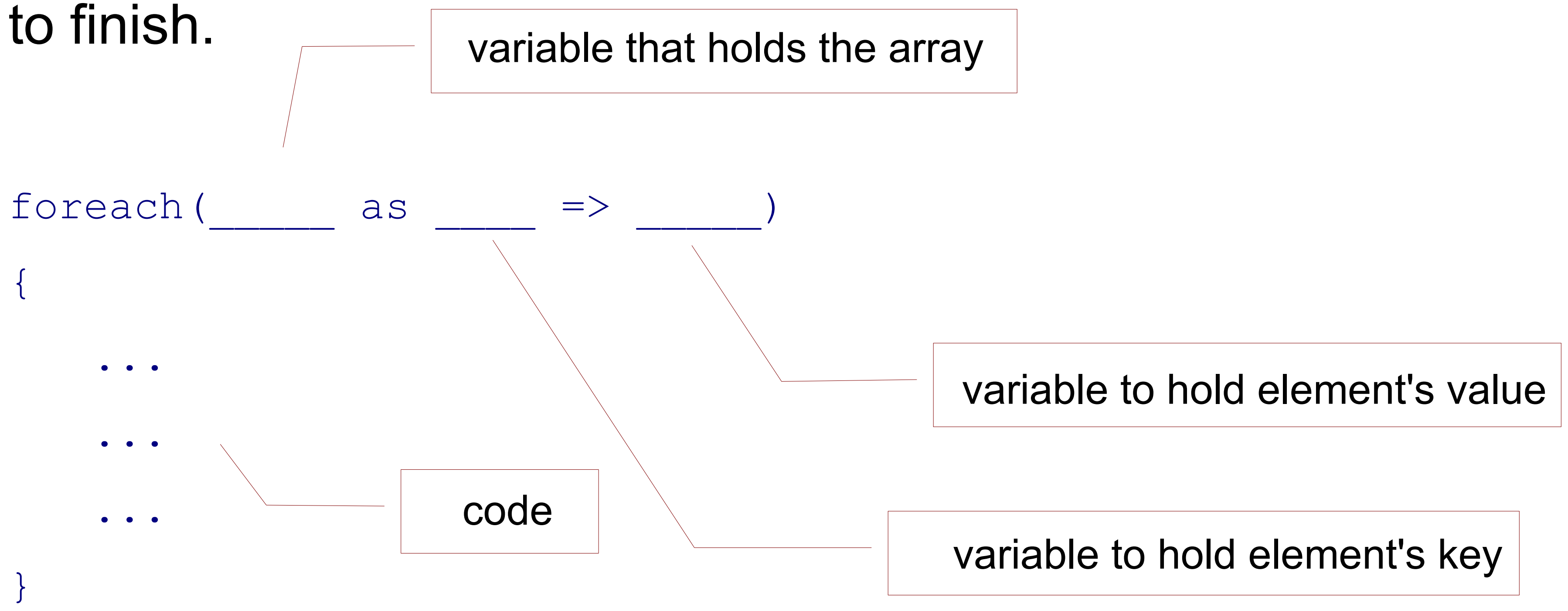
| | |
|---|---|
| `reset()` | resets the pointer to the array initial position. |
| `next()` | moves the pointer to the next element. |
| `prev()` | moves the pointer to the previous element. |
| `current()` | gets the current element's value. |
| `key()` | gets the current element's key. |

## The Array Pointer

```
<?php
$vec = array("a","b","c","d","f","g","h");
reset($vec);
while(key($vec)!==null)
{
    echo key($vec)." is the key and ".current($vec)." is the value<BR>";
    next($vec);
}
?>
```

# The `foreach` Construct

- The foreach construct allows traversing an array from start to finish.



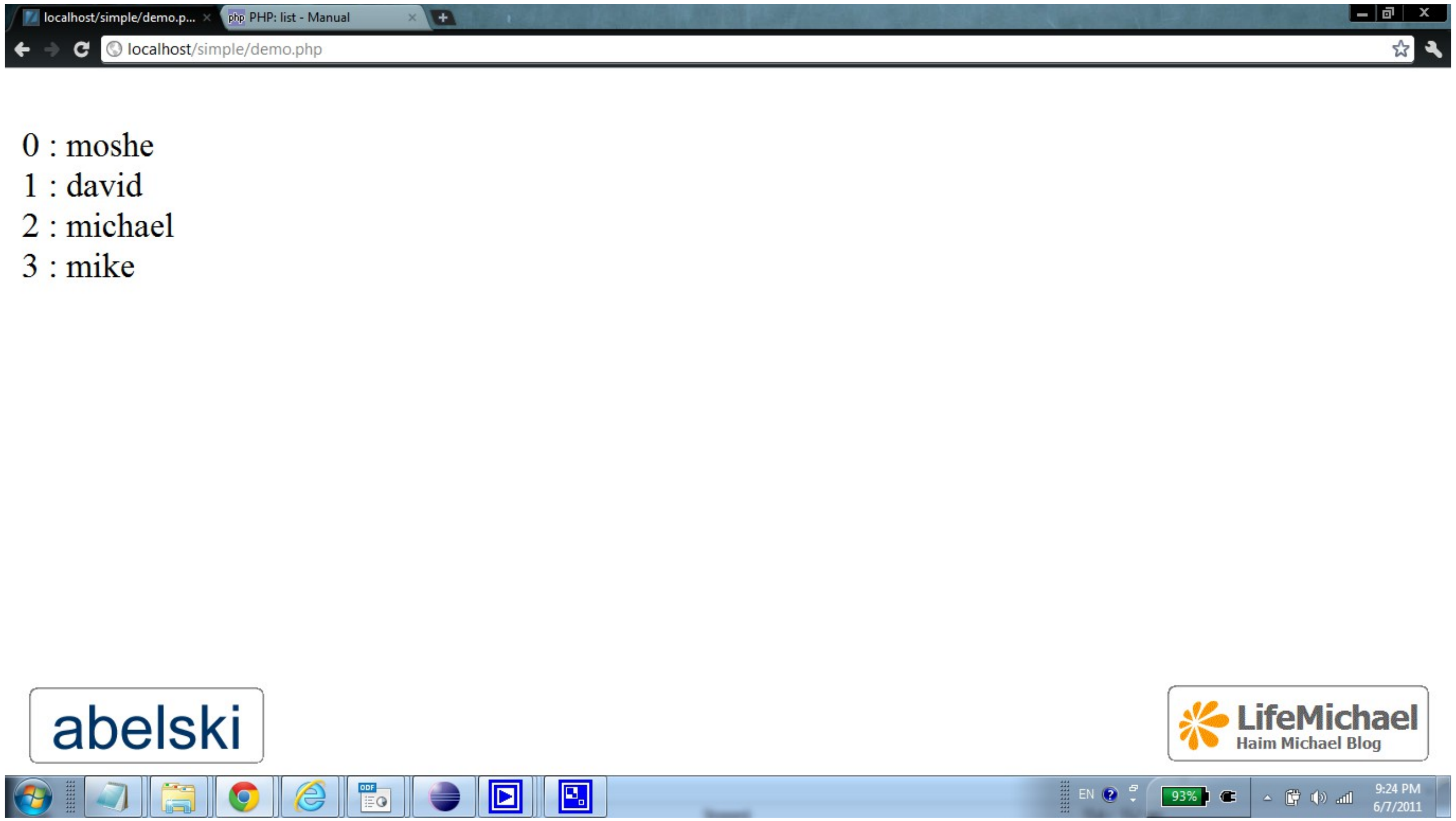
```
foreach(______ as _____ => ______)
{
    ...
    ...
    ...
}
```

variable that holds the array
variable to hold element's value
code
variable to hold element's key

## The `foreach` Construct

```
<?php
$vec = array('moshe','david','michael','mike');
foreach($vec as $key_var => $value_var)
{
    echo "<BR>$key_var : $value_var";
}
?>
```

## The `foreach` Construct



```
0 : moshe
1 : david
2 : michael
3 : mike
```

## The `foreach` Construct

- The following is an alternative syntax for using the foreach construct:

variable that holds the array

```
foreach(______ as _______)
{
    ...
    ...
    ...
}
```

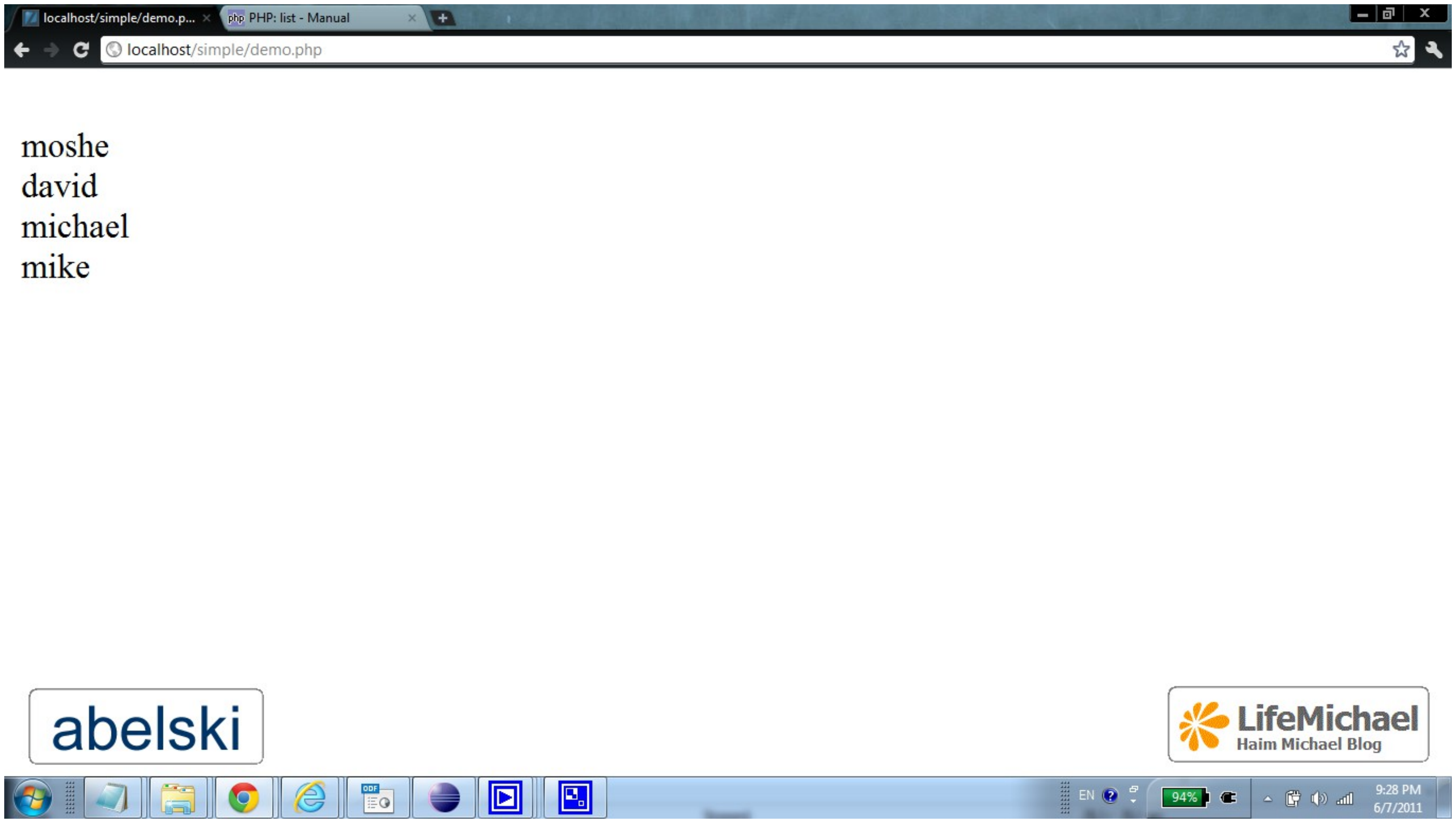
variable to hold element's value

code

## The `foreach` Construct

```
<?php
$vec = array('moshe','david','michael','mike');
foreach($vec as $value_var)
{
    echo "<BR>$value_var";
}
?>
```

# The `foreach` Construct



```
moshe
david
michael
mike
```

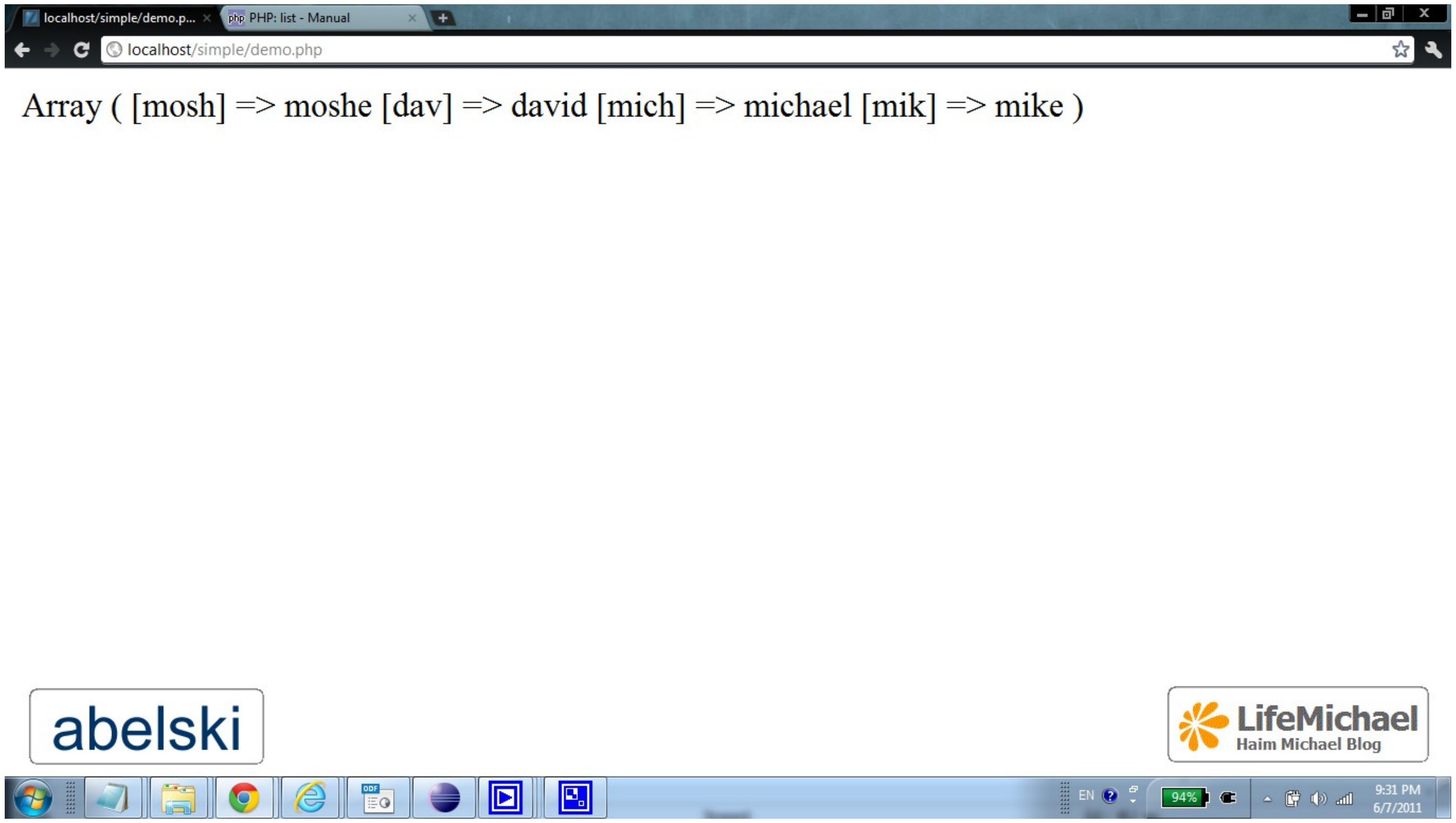
# The `array_combine()` Function

- The `array_combine(array $keys, array $values)` function receives two arrays and creates a new array. The keys are the values of the first array elements. The values are the values of the second array elements.

## The `array_combine()` Function

```
<?php
$values_vec =
array('moshe','david','michael','mike');
$keys_vec = array('mosh','dav','mich','mik');
$vec = array_combine($keys_vec,$values_vec);
print_r($vec);
?>
```

## The `array_combine()` Function



```
Array ( [mosh] => moshe [dav] => david [mich] => michael [mik] => mike )
```

## The `array_walk()` Function

- The `array_walk(array &$vec, callback $function)` function goes over each one of the array's elements and calls the function on each one of them.
- The function should include two parameters. The first is the array's value and the second is the array's key.

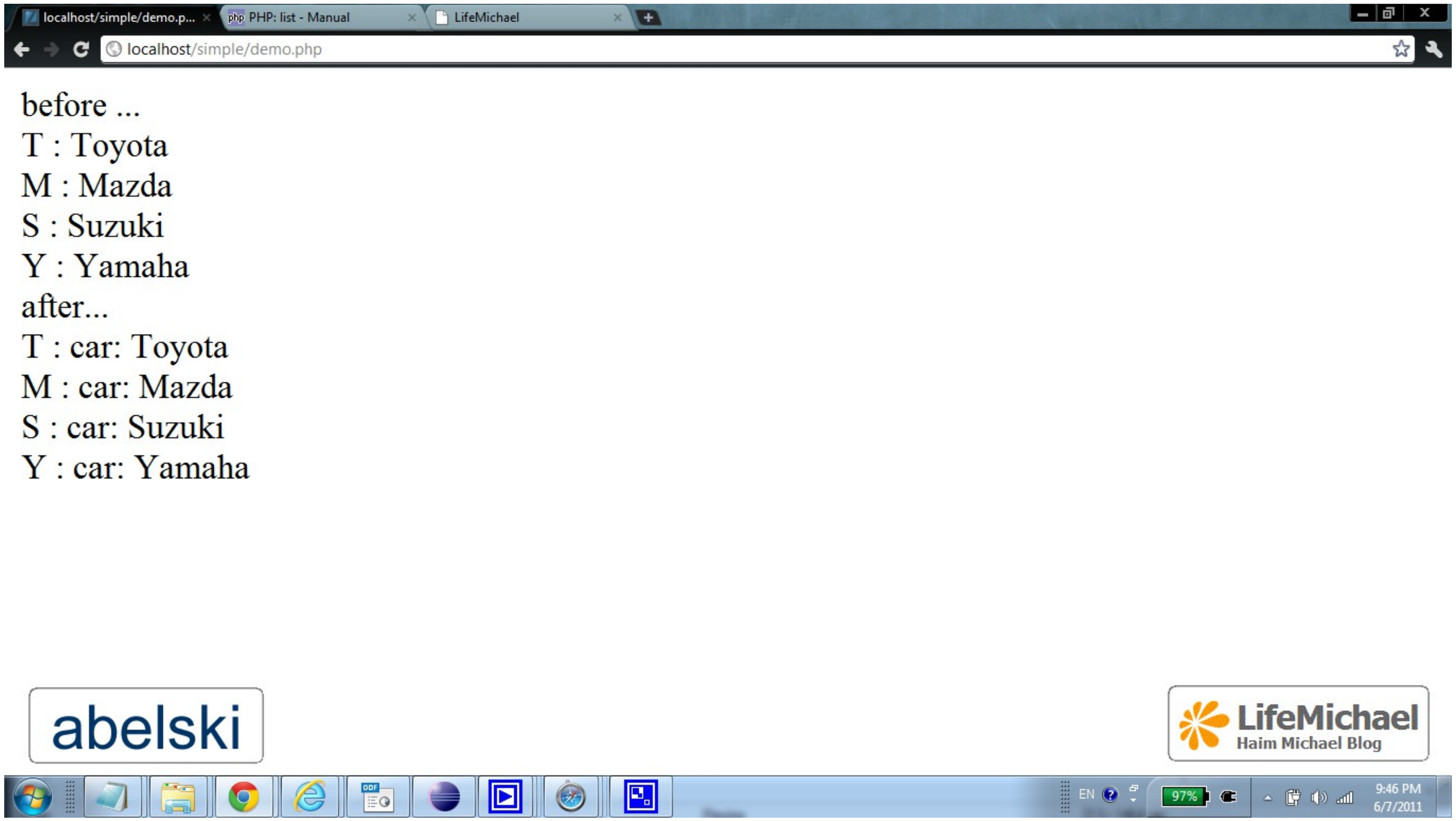
## The `array_walk()` Function

- The `array_walk` has a third optional parameter `($user_data)`. If it is passed then it would be passed as an argument to the function that is called on each one of the elements.

# The `array_walk()` Function

```
<?php

$cars = array("T" => "Toyota", "M" => "Mazda", "S" => "Suzuki", "Y" => "Yamaha");

function changearray(&$val, $key, $prefix)
{
    $val = "$prefix: $val";
}

function printarray($itemvalue, $itemkey)
{
    echo "$itemkey : $itemvalue<br>";
}

echo "before ...<BR>";
array_walk($cars, 'printarray');
array_walk($cars, 'changearray', 'car');
echo "after...<BR>";
array_walk($cars, 'printarray');
?>
```

# The `array_walk()` Function



```
before ...
T : Toyota
M : Mazda
S : Suzuki
Y : Yamaha
after...
T : car: Toyota
M : car: Mazda
S : car: Suzuki
Y : car: Yamaha
```

## The `array_walk_recursive()` Function

- The array_walk_recursive does the same work done by array_walk... with the following improvement: The array_walk_recursive goes over all elements of all arrays that are held as elements of the main array.

## The `array_walk_recursive()` Function

```
<?php
$japan_cars = array("T" => "Toyota", "M" => "Mazda", "S" => "Suzuki", "Y"
=> "Yamaha");
$usa_cars = array("C" => "Chevrolet", "P" => "Pontiac", "C" =>
"Cryzler");
$cars = array("US" => $usa_cars, "JP" => $japan_cars);
function changearray(&$val, $key, $prefix)
{
    $val = "$prefix: $val";
}
function printarray($itemvalue, $itemkey)
{
    echo "$itemkey : $itemvalue<br>";
}
echo "before ...<BR>";
array_walk_recursive($cars, 'printarray');
array_walk_recursive($cars, 'changearray', 'car');
echo "after...<BR>";
array_walk_recursive($cars, 'printarray');
?>
```

# Arrays Sorting

- PHP core functions include various methods for sorting arrays.
- The simplest ones are:

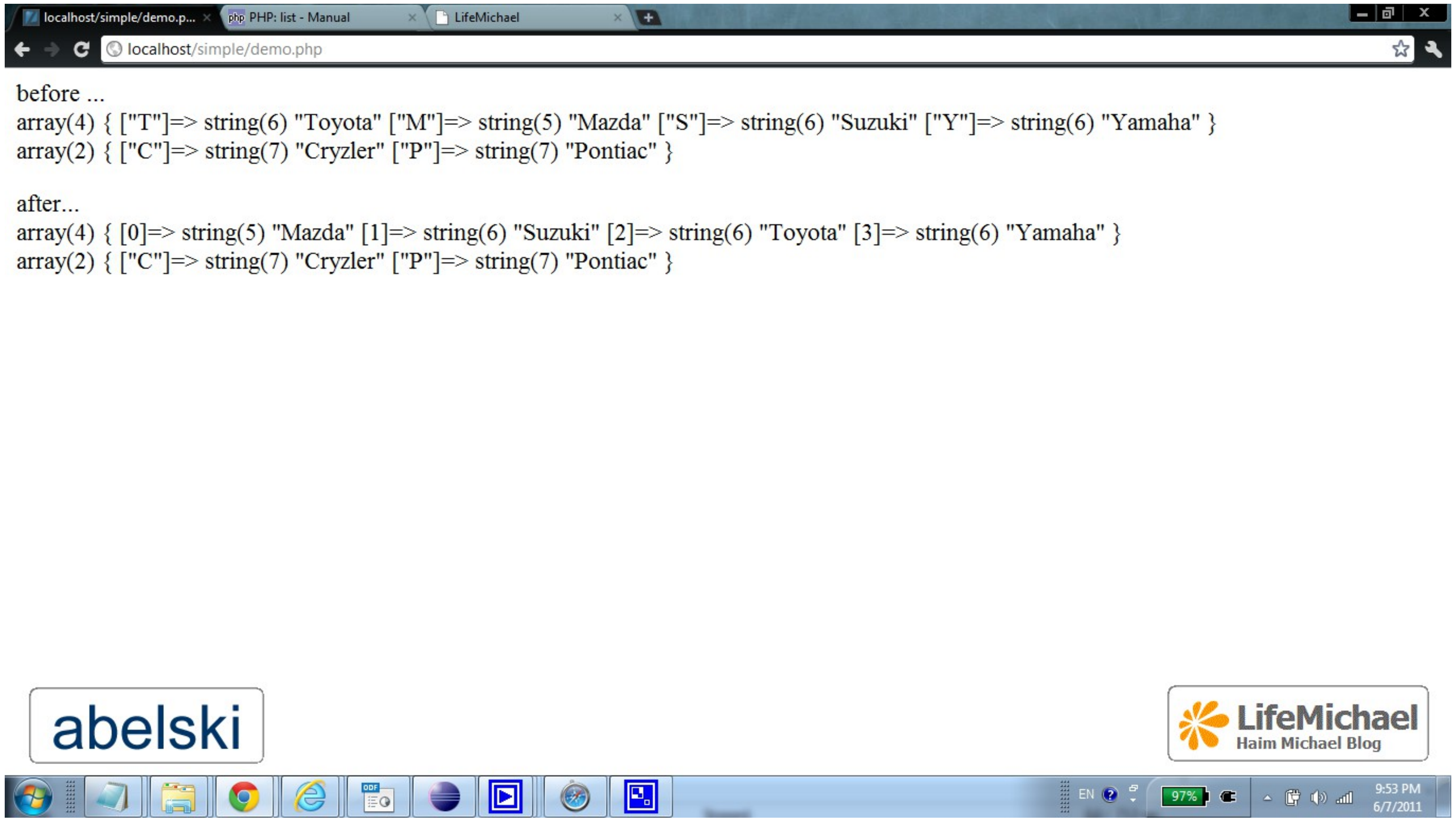
```
sort (array &$vec [, int $sort_flags ])
asort (array &$vec [, int $sort_flags ])
```

- Calling `sort()` destroys all keys and reassign new ones starting from zero. Calling `asort()` keeps the keys unchanged.

# Arrays Sorting

```
<?php
$japan_cars =
array("T" => "Toyota", "M" => "Mazda", "S" => "Suzuki", "Y" => "Yamaha");
$usa_cars =
array("C" => "Chevrolet", "P" => "Pontiac", "C" => "Cryzler");
echo "<P>before ...<BR>";
var_dump($japan_cars);
echo "<BR>";
var_dump($usa_cars);
sort($japan_cars);
asort($usa_cars);
echo "<P>after...<BR>";
var_dump($japan_cars);
echo "<BR>";
var_dump($usa_cars);
?>
```

# Arrays Sorting



```
before ...
array(4) { ["T"]=> string(6) "Toyota" ["M"]=> string(5) "Mazda" ["S"]=> string(6) "Suzuki" ["Y"]=> string(6) "Yamaha" }
array(2) { ["C"]=> string(7) "Cryzler" ["P"]=> string(7) "Pontiac" }

after...
array(4) { [0]=> string(5) "Mazda" [1]=> string(6) "Suzuki" [2]=> string(6) "Toyota" [3]=> string(6) "Yamaha" }
array(2) { ["C"]=> string(7) "Cryzler" ["P"]=> string(7) "Pontiac" }
```

# Arrays Sorting

- Both `sort()` and `asort()` allows passing a second optional parameter, that configures the operation. This second optional parameter can be one of the following possibilities:

  SOFT_REGULAR

  This is the default. Sorting will be performed according to elements' values and without introducing any change.

  SORT_NUMERIC

  Each element's value will be first converted into a numeric value. The sorting will be according to these numeric values.

  SORT_STRING

  Sorting will be according to the elements' values converted into strings.

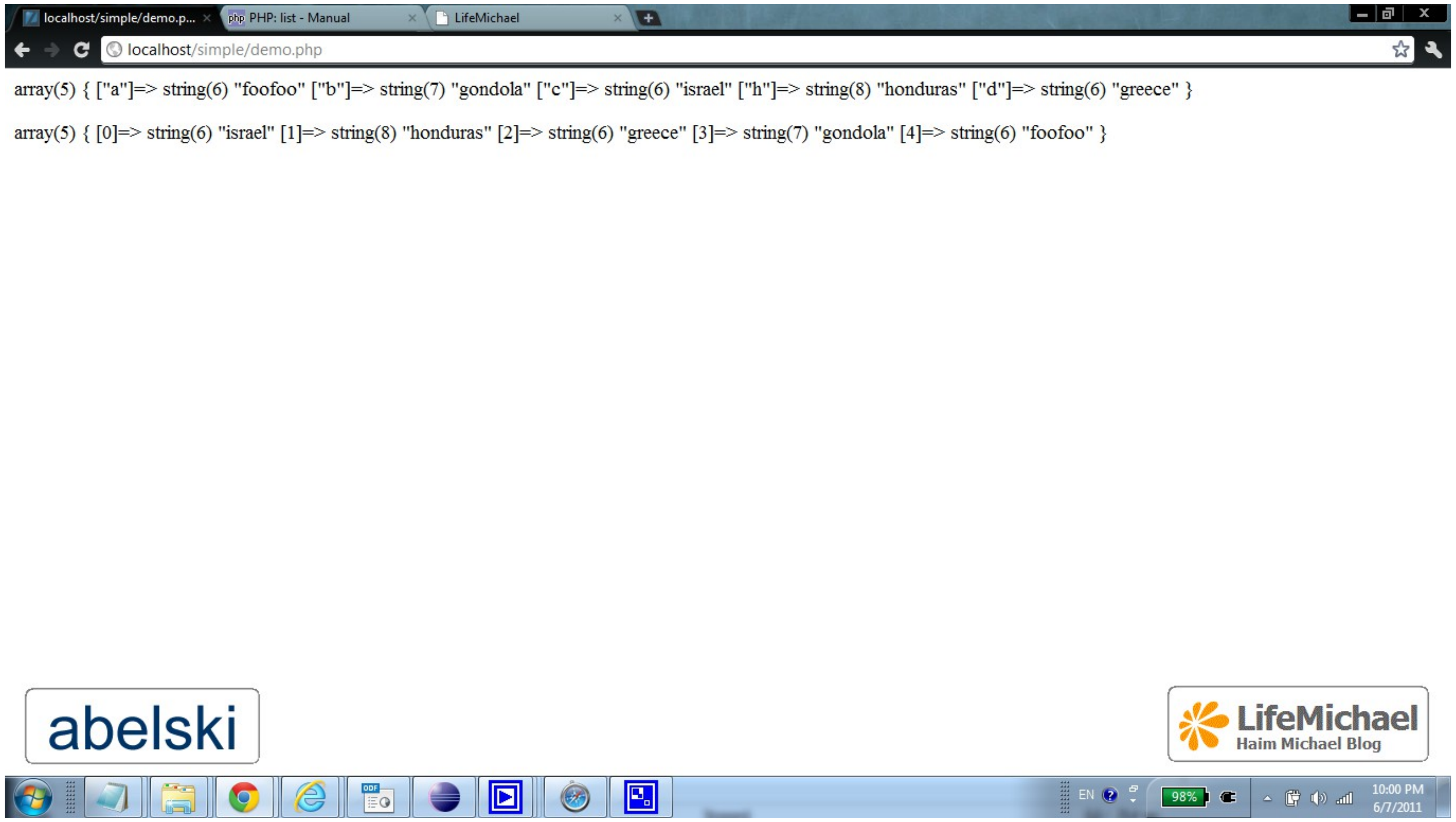
## Arrays Sorting

- The `rsort()` function sorts an array in a reverse order.
- The `rsort()` function removes all elements' keys and assign new ones.

## Arrays Sorting

```
<?php
$vec = array(
    "a"=>"foofoo",
    "b"=>"gondola",
    "c"=>"israel",
    "h"=>"honduras",
    "d"=>"greece");
var_dump($vec);
rsort($vec);
echo "<p>";
var_dump($vec);
?>
```

# Arrays Sorting


localhost/simple/demo.p...
PHP: list - Manual
LifeMichael
localhost/simple/demo.php

```
array(5) { ["a"]=> string(6) "foofoo" ["b"]=> string(7) "gondola" ["c"]=> string(6) "israel" ["h"]=> string(8) "honduras" ["d"]=> string(6) "greece" }

array(5) { [0]=> string(6) "israel" [1]=> string(8) "honduras" [2]=> string(6) "greece" [3]=> string(7) "gondola" [4]=> string(6) "foofoo" }
```

## Arrays Sorting

- The `ksort()` and `krsort()` functions sort an array by its elements' keys.

## Arrays Sorting

- The `usort(array &$vec, callback $function)` function sorts an array by its elements' values and using a user defined comparison function.

## Arrays Sorting

```php
<?php
class Student
{
    private $id;
    private $average;
    private $name;

    function __construct($idVal, $averageVal, $nameVal)
    {
        $this->id = $idVal;
        $this->average = $averageVal;
        $this->name = $nameVal;
    }

    public function getId()
    {
        return $this->id;
    }

    public function getAverage()
    {
        return $this->average;
    }
```

## Arrays Sorting

```
    public function getName()
    {
        return $this->name;
    }

    public function __toString()
    {
        return $this->getName () . " id=" . $this->getId () .
            " average=" . $this->getAverage ();
    }
}
```

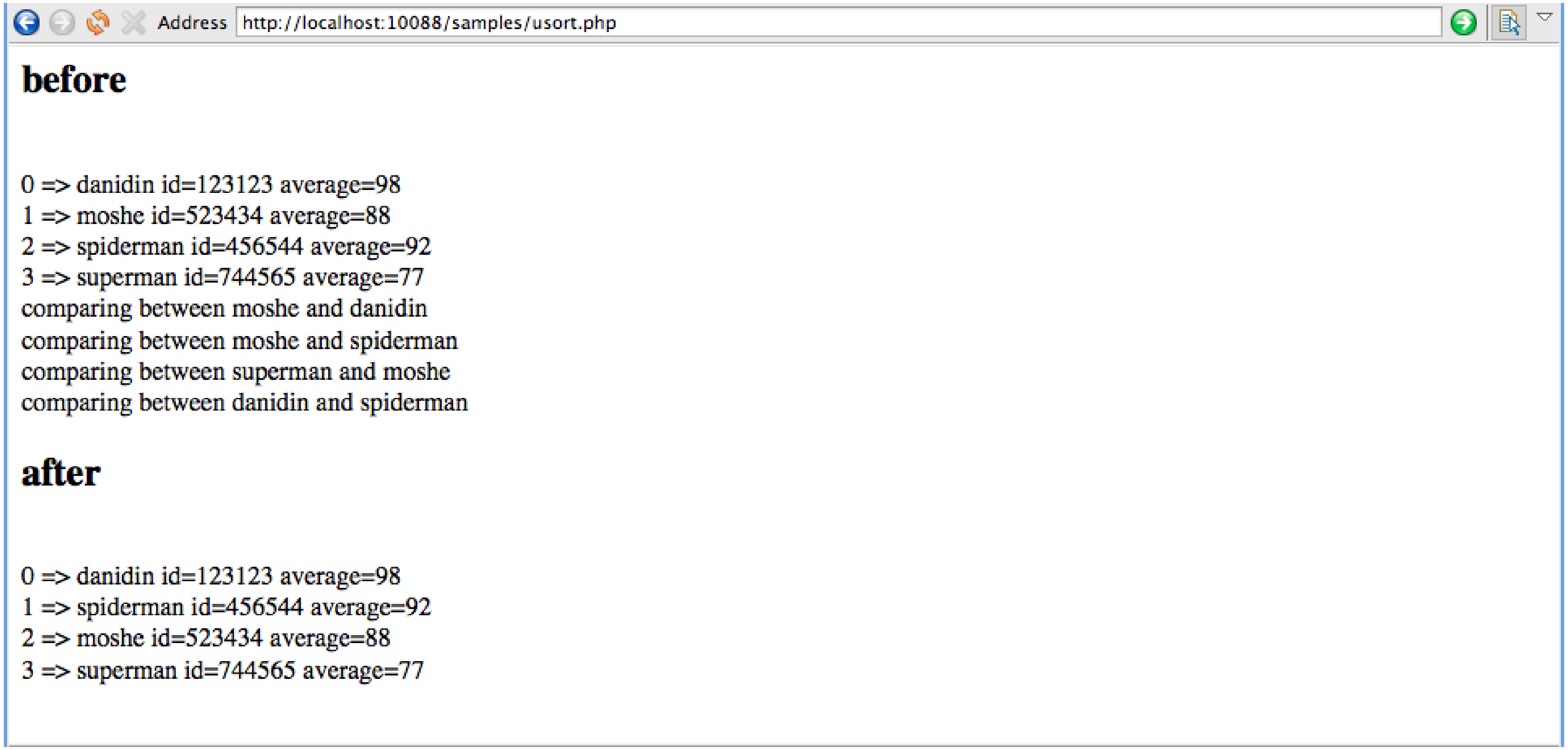
## Arrays Sorting

```
$vec = [
        new Student ( 123123, 98, "danidin" ),
        new Student ( 523434, 88, "moshe" ),
        new Student ( 456544, 92, "spiderman" ),
        new Student ( 744565, 77, "superman" )
];

echo "<h2>before</h2>";
foreach ( $vec as $k => $v )
{
    echo "<Br>$k => " . $v;
}
```

## Arrays Sorting

```
usort ( $vec, function ($a, $b)
{
    echo "<br>comparing between ".$a->getName()." and ".$b->getName();
    return $a->getId() - $b->getId();
} );

echo "<h2>after</h2>";
foreach ( $vec as $k => $v )
{
    echo "<Br>$k => " . $v;
}

?>
```

## Arrays Sorting



```
before

0 => danidin id=123123 average=98
1 => moshe id=523434 average=88
2 => spiderman id=456544 average=92
3 => superman id=744565 average=77
comparing between moshe and danidin
comparing between moshe and spiderman
comparing between superman and moshe
comparing between danidin and spiderman

after

0 => danidin id=123123 average=98
1 => spiderman id=456544 average=92
2 => moshe id=523434 average=88
3 => superman id=744565 average=77
```

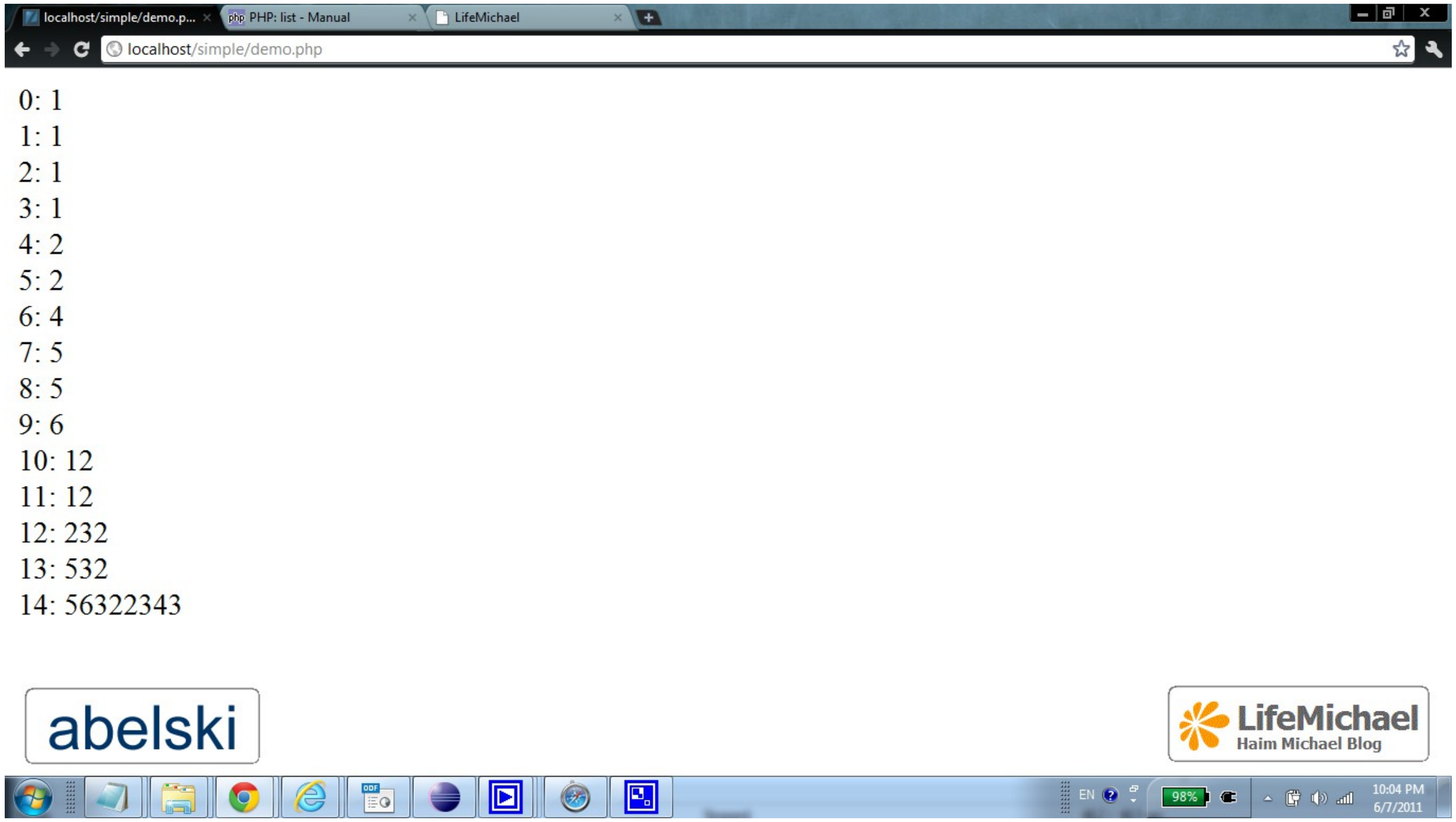
## Arrays Sorting

```
<?php
function cmp($a, $b)
{
    if ($a == $b) {
        return 0;
    }
    return ($a < $b) ? -1 : 1;
}

$vec = array(12,532,12,56322343,232,5,2,1,1,1,4, 2, 5, 6, 1);

usort($vec, "cmp");

foreach ($vec as $key => $value)
{
    echo "$key: $value<BR>";
}
?>
```

## Arrays Sorting

```
<?php
function cmp($a, $b)
{
    if ($a == $b) {
        return 0;
    }
    return ($a < $b) ? -1 : 1;
}

$vec = array(12,532,12,56322343,232,5,2,1,1,1,4, 2, 5, 6, 1);

usort($vec, "cmp");

foreach ($vec as $key => $value)
{
    echo "$key: $value<BR>";
}
?>
```

# Arrays Sorting



```
0: 1
1: 1
2: 1
3: 1
4: 2
5: 2
6: 4
7: 5
8: 5
9: 6
10: 12
11: 12
12: 232
13: 532
14: 56322343
```

## Arrays Shuffle

- Calling the `shuffle()` function will scramble arrays' elements in a randomize order.

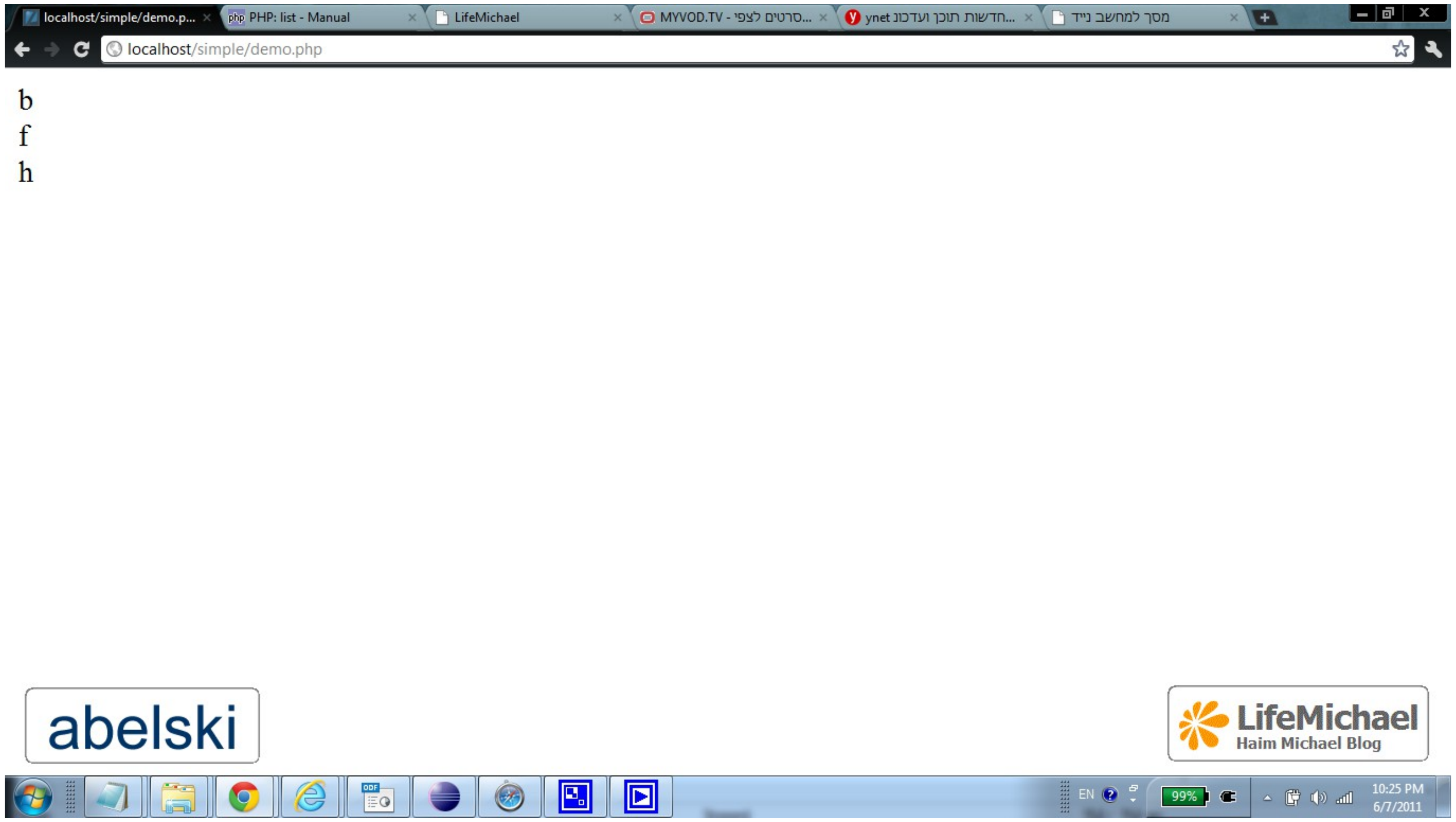
## Arrays Randomized Elements

- Using the `array_rand(array $input [, int $num_req ] )` function we can get randomized selected elements from our array. The first parameter is our array. The second parameter is the number of elements we request.
- If we request one element only, `array_rand()` returns the key for the random element. If we request more than one element, `array_rand()` returns an array of keys for the random elements.

## Arrays Randomized Elements

```php
<?php
$vec = array("a","b","c","d","f","g","h");
$random_keys = array_rand($vec,3);
echo $vec[$random_keys[0]];
echo "<BR>";
echo $vec[$random_keys[1]];
echo "<BR>";
echo $vec[$random_keys[2]];
echo "<BR>";
?>
```

# Arrays Randomized Elements



```
b
f
h
```

## Arrays as Stacks

- The `array_push()` and `array_pop()` functions enable us to use an array as a stack.

```
int array_push ( array &$array , mixed $var [, mixed $... ] )
```

This function pushes the passed values onto the end of the array. The array's length is increased by the number of the passed variables. This functions returns the number of elements, the array has.

```
mixed array_pop ( array &$array )
```

This function returns the last value of the array and shorten its length by one.

# Arrays as Sets

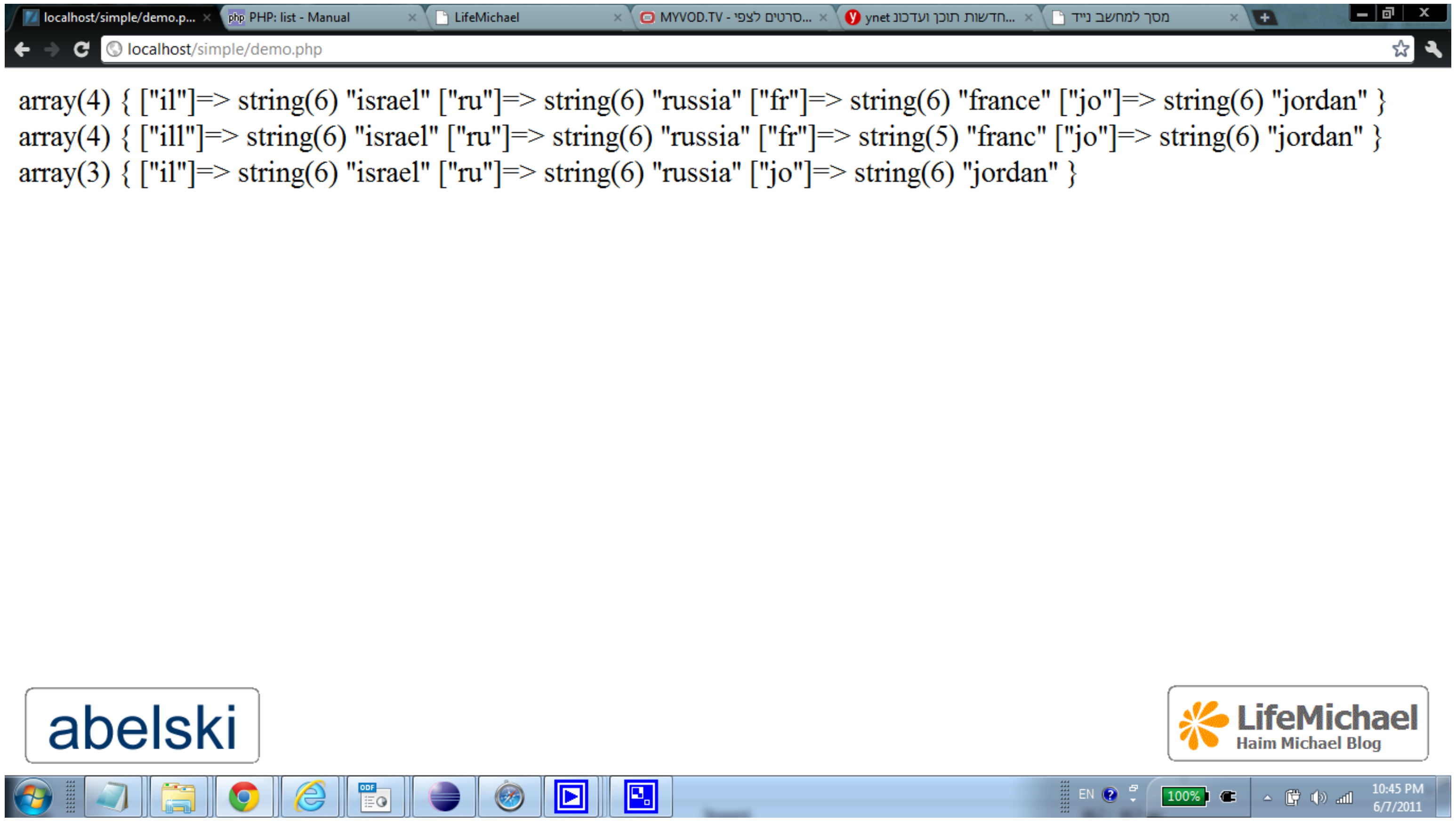
- The `array_intersect()` function returns an array containing all the values of array1 that are present in all other arrays. The keys are preserved.

```
array array_intersect ( array $array1 , array $array2
    [, array $ ... ] )
```

# Arrays as Sets

```php
<?php
$vecA = array("il"=>"israel","ru"=>"russia","fr"=>"france","jo"=>"jordan");
var_dump($vecA);
echo "<br/>";
$vecB = array("ill"=>"israel","ru"=>"russia","fr"=>"franc","jo"=>"jordan");
var_dump($vecB);
echo "<br/>";
$vecC = array_intersect($vecA,$vecB);
var_dump($vecC);
?>
```

## Arrays as Sets



```
array(4) { ["il"]=> string(6) "israel" ["ru"]=> string(6) "russia" ["fr"]=> string(6) "france" ["jo"]=> string(6) "jordan" }
array(4) { ["ill"]=> string(6) "israel" ["ru"]=> string(6) "russia" ["fr"]=> string(5) "franc" ["jo"]=> string(6) "jordan" }
array(3) { ["il"]=> string(6) "israel" ["ru"]=> string(6) "russia" ["jo"]=> string(6) "jordan" }
```

# Arrays Shorter Syntax

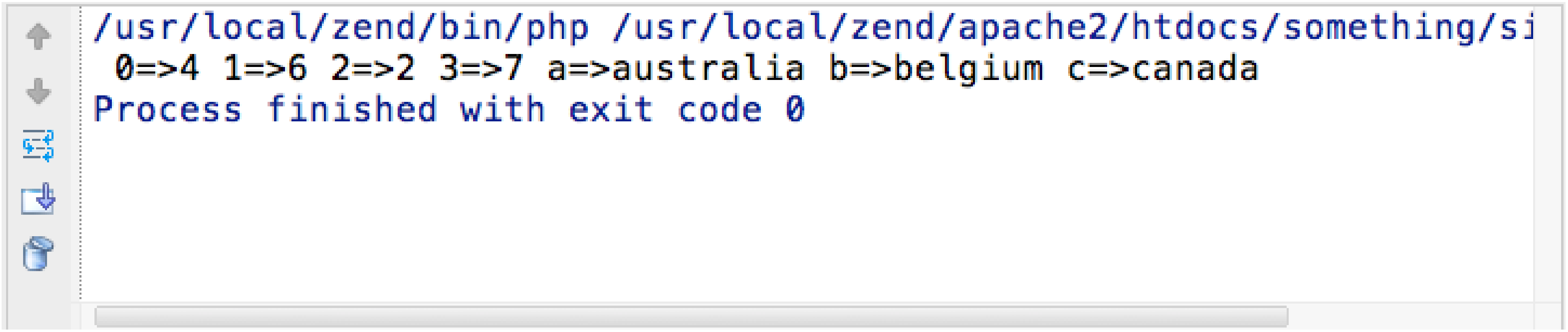
- As of PHP 5.4 we can create new arrays in the following new short syntax:

```
$vec = [34,234,75,4];
```

# Arrays Shorter Syntax

```
<?php
$vec_a = [4,6,2,7];
$vec_b = ['a'=>'australia','b'=>'belgium','c'=>'canada'];
foreach($vec_a as $k=>$v)
{
    echo " ".$k."=>".$v;
}
foreach($vec_b as $k=>$v)
{
    echo " ".$k."=>".$v;
}
```

## Arrays Shorter Syntax



```
/usr/local/zend/bin/php /usr/local/zend/apache2/htdocs/something/si
 0=>4 1=>6 2=>2 3=>7 a=>australia b=>belgium c=>canada
Process finished with exit code 0
```

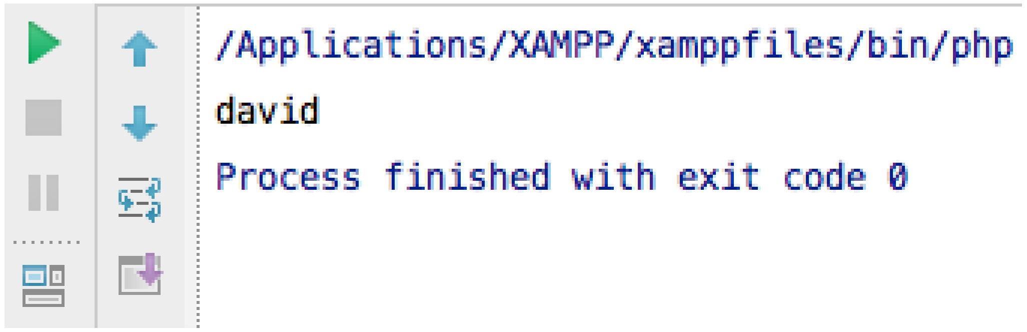
## Array Dereferencing

- As of PHP 5.5 it is possible to dereference the array directly.

# Array Dereferencing

```
<?php
echo ["david","anat","limor","ilana"][0];
?>
```

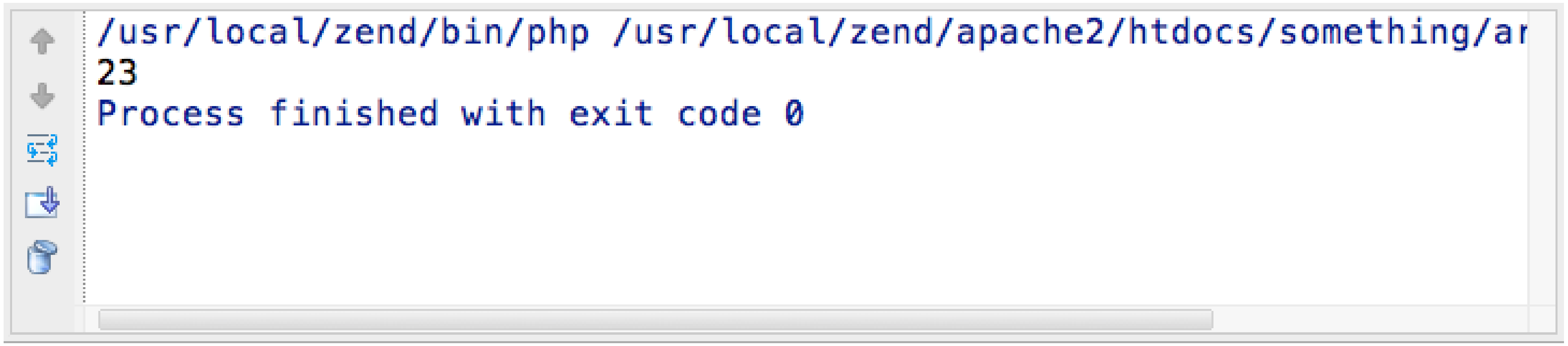
## Array Dereferencing



```
/Applications/XAMPP/xamppfiles/bin/php
david
Process finished with exit code 0
```

The Output

# Function Array Dereferencing



```
/usr/local/zend/bin/php /usr/local/zend/apache2/htdocs/something/ar
23
Process finished with exit code 0
```

## Array Dereferencing

- As of PHP 5.5 we can develop a function that returns an array and use a call to that function as if it was an array.

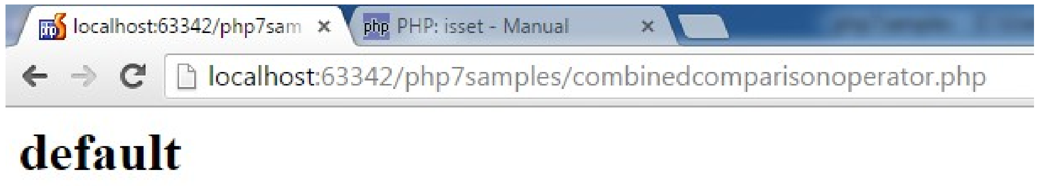
## The `??` Operator

- The `??` operator, that was introduced in PHP 7, is also known as the isset ternary operator, is a shorthand notation for performing `isset()` checks in the ternary operator.
- This new operator assists us with those cases in which we need to check whether the array we work with has a specific key so we could use its value and if not, then another value will be used instead.

## The `??` Operator

```
$vec = ['a'=>'abba','b'=>'baba','m'=>'mama'];

//before PHP7
//$temp = isset($vec['d'])?$vec['d']:'default';

$temp = $vec['d']??'default';

echo "<h1>$temp</h1>";
```

## The ?? Operator



localhost:63342/php7sam ×

PHP: isset - Manual ×

localhost:63342/php7samples/combinedcomparisonoperator.php

```
default
```

## Array Constants

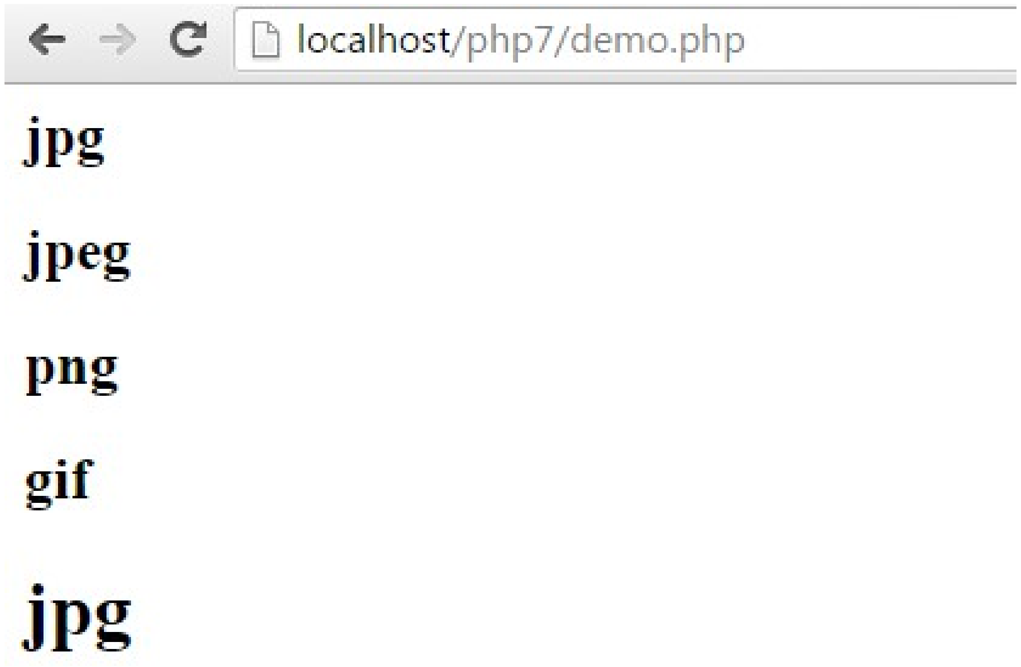
- PHP 5.6 added the possibility to define array constants using the const keyword. As of PHP 7 we can define array constants using the `define()` function.

## Array Constants

```
<?php
define('IMAGE_TYPES', ['jpg', 'jpeg', 'png', 'gif']);

foreach(IMAGE_TYPES as $v) {
    echo "<h2>".$v."</h2>";
}

echo "<h1>".IMAGE_TYPES[0]."</h1>";
?>
```

YouTube

# Array Constants



localhost/php7/demo.php

```
jpg

jpeg

png

gif

jpg
```