The Standard PHP Library

Introduction

- The standard PHP library (SPL) includes a collection of interfaces and classes and it aims at assisting us coping with standard problems.

Data Structures

- SPL provides us with standard data structures, including double linked lists (`SplDoublyLinkedList`, `SplStack` and `SplQueue`), heaps (`SplHeap`, `SplMaxHeap`, `SplMinHeap` and `SplPriorityQueue`), an array (`SplFixedArray`) and a map (`SplObjectsStorage`).

Iterators

- SPL provides us with a set of iterator classes, such as `AppendIterator`, `ArrayIterator`, `CachingIterator`, `DirectoryIterator` and many others.

Interfaces

❖ SPL provides us with a set of interfaces we can implement in the classes we define.

The `Countable` Interface

❖ Implementing this interface in our class we will be able to use the `count()` function, passing over a reference for an object instantiated from our class.

```
interface Countable
{
    function count();
}
```

The `Countable` Interface

```php
<?php
class Library implements Countable
{
    var $books = array();
    function addBook(Book $obVal)
    {
        $this->books[$obVal->getId()]=$obVal;
    }
    public function count()
    {
        $num = count($this->books);
        return $num;
    }
}
```

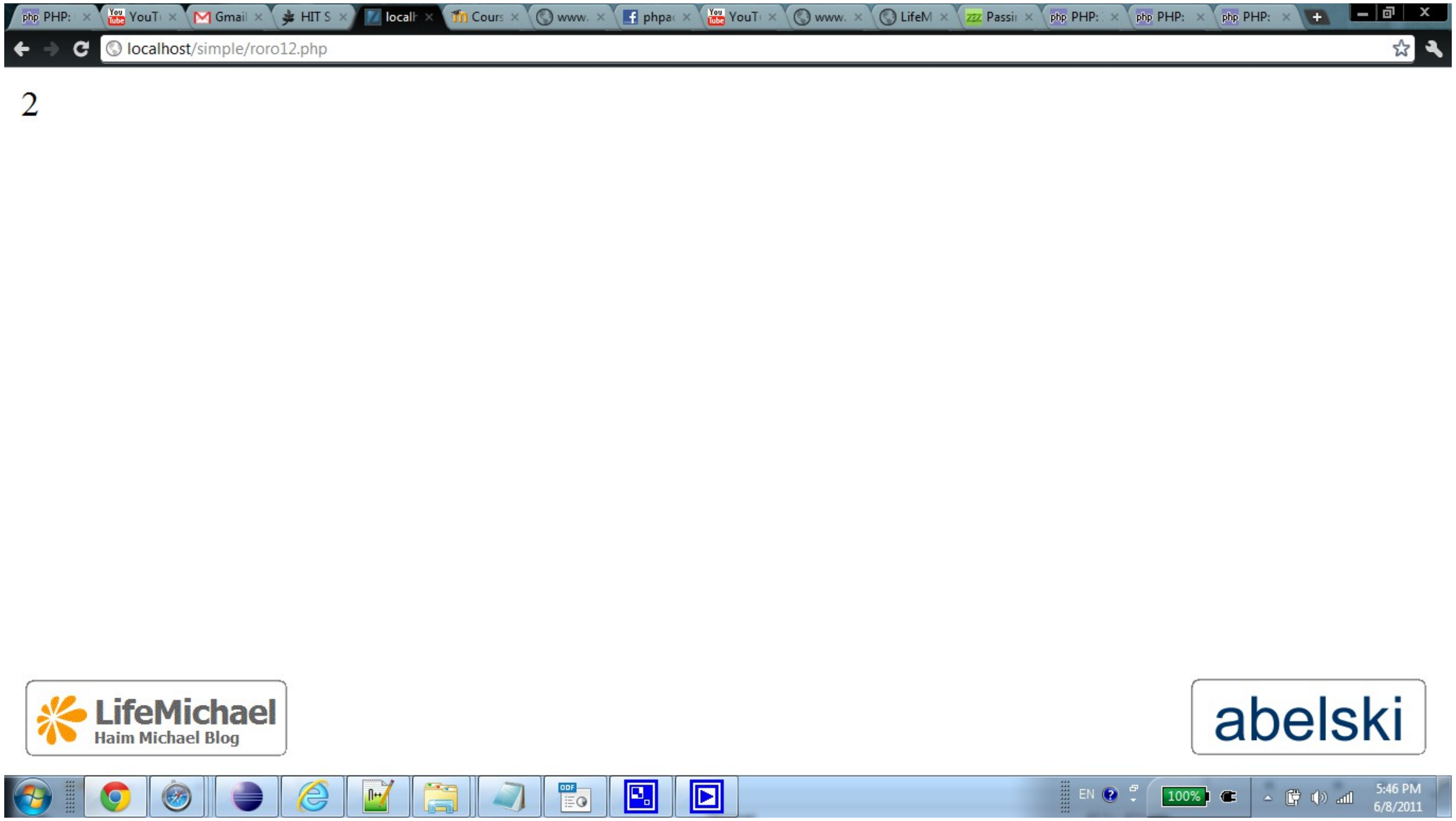
The `Countable` Interface

```
class Book
{
    private $name;
    private $id;
    function __construct($namVal,$iVal)
    {
        $this->name = $namVal;
        $this->id = $iVal;
    }
    function getId()
    {
        return $this->id;
    }
    function getName()
    {
        return $this->name;
    }
}
```

The `Countable` Interface

```
$ob = new Library();
$ob->addBook(new Book("scala basics",123));
$ob->addBook(new Book("php fundamental",532));
echo count($ob);
?>
```

The `Countable` Interface



```
2
```

The `ArrayAccess` Interface

- Implementing this interface in our class will enable us to instantiate our class and use the new object as if it was an array.

```
interface ArrayAccess
{
    function offsetSet($offset, $value);
    function offsetGet($offset);
    function offsetUnset($offset);
    function offsetExists($offset);
}
```

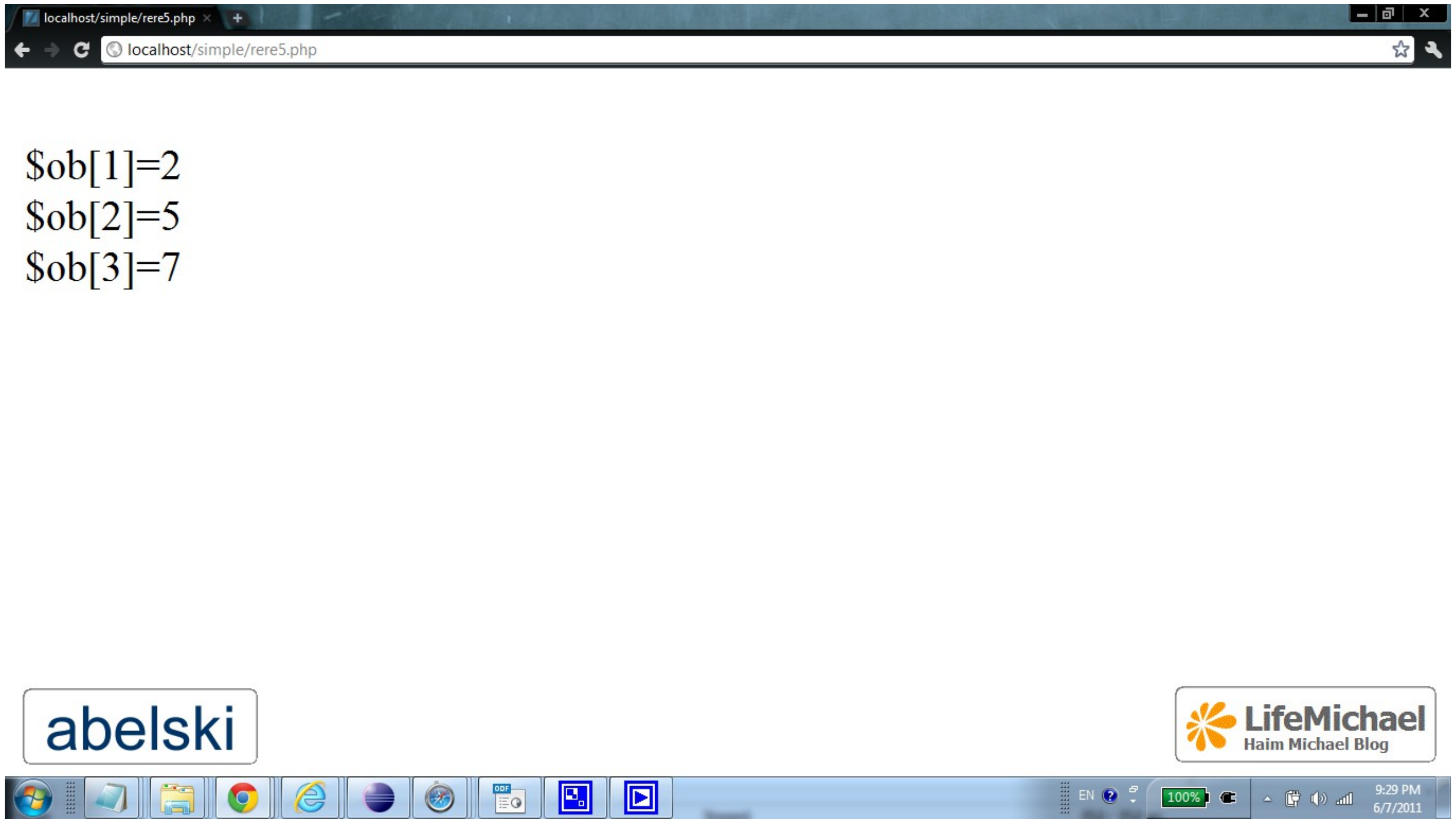
The `ArrayAccess` Sample

```
<?php

class MyVector implements ArrayAccess
{
    private $vec = array();
    function offsetSet ($index, $value)
    {
        if (!is_numeric ($index))
        {
            throw new Exception ("Invalid key");
        }
        $this->vec[$index] = $value;
    }
    function offsetGet ($index)
    {
        return $this->vec[$index];
    }
    function offsetUnset ($index)
    {
        unset ($this->vec[$index]);
    }
    function offsetExists ($index)
    {
        return array_key_exists ($this->vec, $index);
    }
}
```

The `ArrayAccess` Sample

```
$ob = new MyVector();
$ob[1] = 2;
$ob[2] = $ob[1] + 3;
$ob[3] = $ob[1] + $ob[2];

printf("<BR>\$ob[1]=%d",$ob[1]);
printf("<BR>\$ob[2]=%d",$ob[2]);
printf("<BR>\$ob[3]=%d",$ob[3]);

?>
```

The `ArrayAccess` Sample



```
$ob[1]=2
$ob[2]=5
$ob[3]=7
```

The `Iterator` Interface

❖ Implementing this interface in our class will enable us to instantiate our class and get an iterator for the new object.

```
interface Iterator
{
    function current();
    function next();
    function rewind();
    function key();
    function valid();
    function seek($key);
}
```

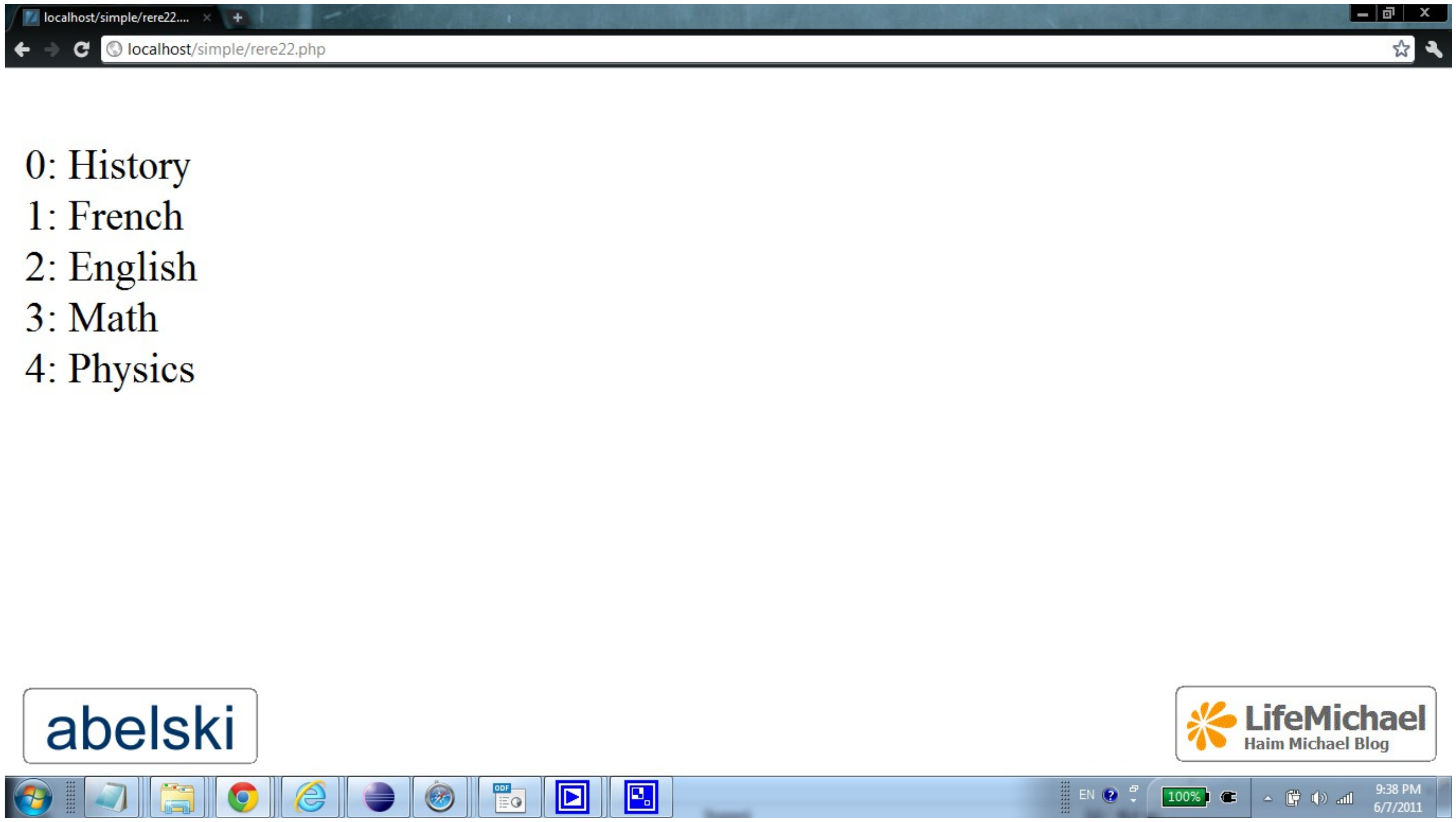
The `Iterator` Sample

```
<?php

class student implements iterator
{
    private $courses = array("History","French","English","Math","Physics");

    private $index;

    function current()
    {
        return $this->courses[$this->index];
    }

    function next()
    {
        $this->index+=1;
    }
```

The `Iterator` Sample

```
    function rewind()
    {
        $this->index=0;
    }

    function key()
    {
        return $this->index;
    }

    function valid()
    {
        return isset($this->courses[$this->index]);
    }
}

$ob = new student();

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

The `Iterator` Sample



```
0: History
1: French
2: English
3: Math
4: Physics
```

Functions

- SPL provides us with more than a few useful functions we can use in our code.

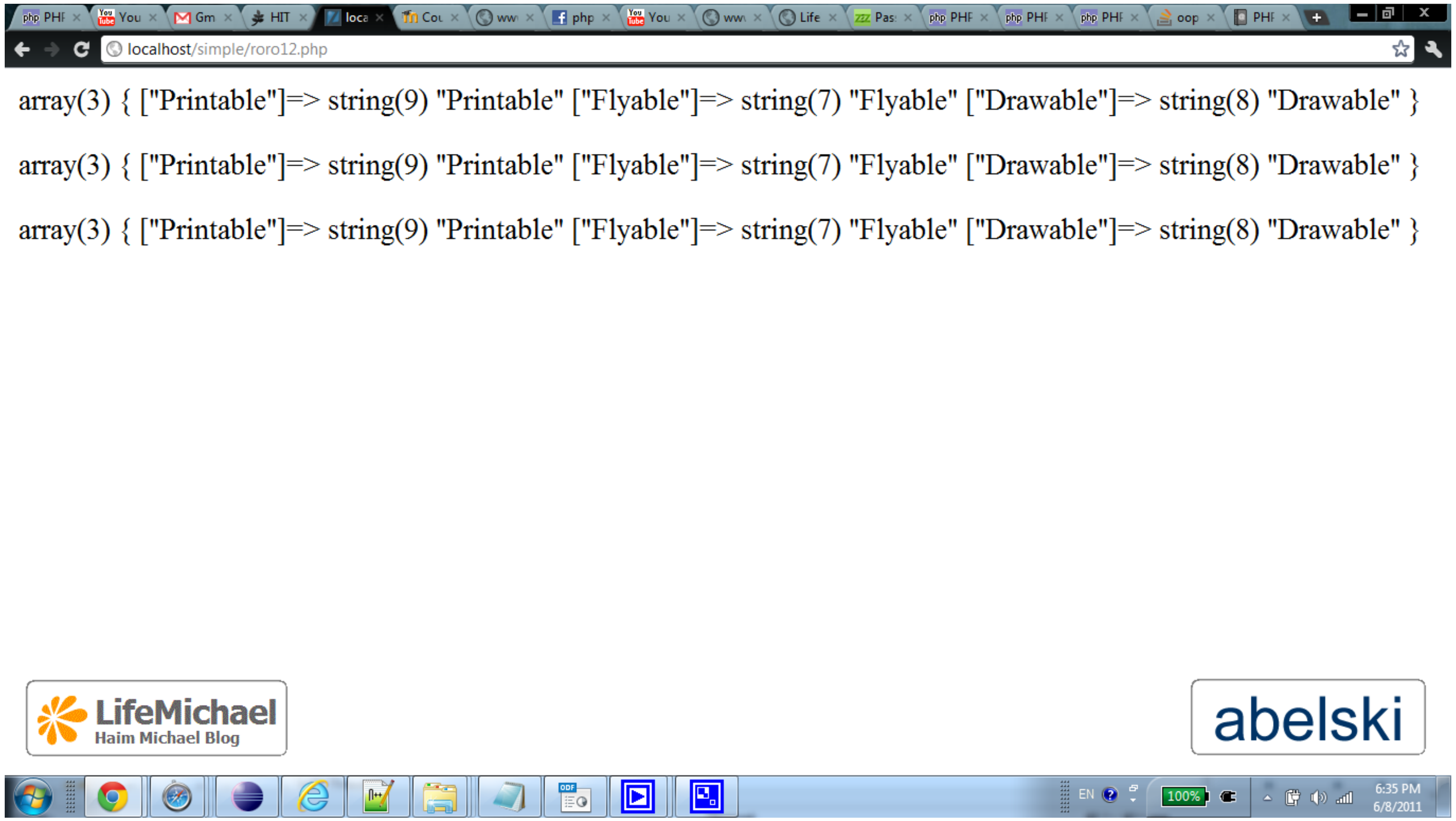
The `class_implements` Function

- This function returns the an array with the names of the interfaces implemented by the class we pass over its name.

The `class_implements` Function

```
<?php

interface Printable { }
interface Flyable { }
interface Drawable { }

class Box implements Printable, Flyable, Drawable {}

var_dump(class_implements(new Box()));
echo "<p>";
var_dump(class_implements('Box'));
echo "<p>";
var_dump(class_implements(Box));

?>
```

The `class_implements` Function



```
array(3) { ["Printable"]=> string(9) "Printable" ["Flyable"]=> string(7) "Flyable" ["Drawable"]=> string(8) "Drawable" }

array(3) { ["Printable"]=> string(9) "Printable" ["Flyable"]=> string(7) "Flyable" ["Drawable"]=> string(8) "Drawable" }

array(3) { ["Printable"]=> string(9) "Printable" ["Flyable"]=> string(7) "Flyable" ["Drawable"]=> string(8) "Drawable" }
```

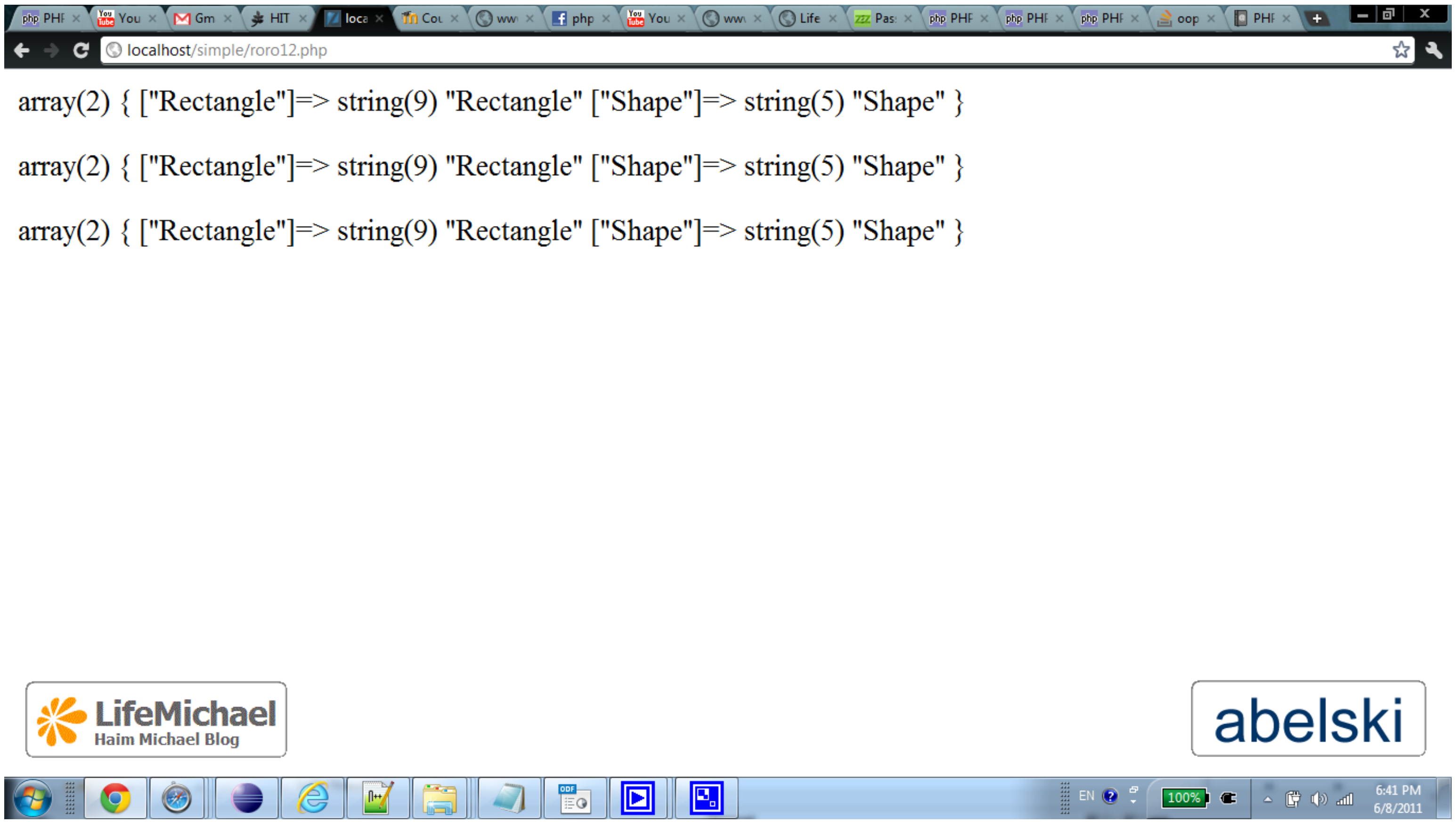
The `class_parents` Function

- This function returns an array with the names of all parent classes our class inherits either directly or indirectly.

The `class_parents` Function

```
<?php

class Shape { }
class Rectangle extends Shape { }
class Box extends Rectangle { }

var_dump(class_parents(new Box()));
echo "<p>";
var_dump(class_parents('Box'));
echo "<p>";
var_dump(class_parents(Box));

?>
```

The `class_parents` Function


localhost/simple/roro12.php

```
array(2) { ["Rectangle"]=> string(9) "Rectangle" ["Shape"]=> string(5) "Shape" }

array(2) { ["Rectangle"]=> string(9) "Rectangle" ["Shape"]=> string(5) "Shape" }

array(2) { ["Rectangle"]=> string(9) "Rectangle" ["Shape"]=> string(5) "Shape" }
```